DOT/FAA/TC-23/06

Federal Aviation Administration
William J. Hughes Technical Center
Aviation Research Division
Atlantic City International Airport
New Jersey 08405

Assessing the Use of Machine Learning to Find the Worst-Case Execution Time of Avionics Software

April 2023

Final report

U.S. Department of Transportation
Federal Aviation Administration

NOTICE

This document is disseminated under the sponsorship of the U.S. Department of Transportation in the interest of information exchange. The U.S. Government assumes no liability for the contents or use thereof. The U.S. Government does not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because they are considered essential to the objective of this report. The findings and conclusions in this report are those of the author(s) and do not necessarily represent the views of the funding agency. This document does not constitute FAA policy. Consult the FAA sponsoring organization listed on the Technical Documentation page as to its use.

This material is based upon work funded and supported by the Department of Transportation under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.
This report was prepared for the SEI Administrative Agent AFLCMC/AZS 5 Eglin Street Hanscom AFB, MA 01731-2100

[DISTRIBUTION STATEMENT B] Distribution authorized to U.S. Government Agencies only (release would constitute premature dissemination) (determination date: 2019-06-30). Other requests for this document shall be referred to Dr. Srini Mandalapu, FAA William J. Hughes Technical Center, Atlantic City International Airport, Egg Harbor Township, NJ 08405.
DM22-1197

This report is available at the Federal Aviation Administration William J. Hughes Technical Center's Full-Text Technical Reports page: actlibrary.tc.faa.gov in Adobe Acrobat portable document format (PDF).

Contents

Figures

Tables

Acronyms

Acronym	Definition
AI	Artificial intelligence
ANN	Artificial neural network
BCET	Best-case execution time
CFG	Control flow graph
FAA	Federal Aviation Administration
FFT	Fast Fourier transform
FIR	Finite impulse response
ML	Machine learning
RRT	Rapidly-exploring random tree
SFP	Single feasible path
TQL	Tool qualification level
TLB	Translation lookaside buffer
WCET	Worst-case execution time

Executive summary

Many parts in aircraft today rely on software that interacts with its physical environment. Typically, this interaction involves taking sensor readings, sending actuation commands, reading commands from humans, and presenting information to humans. These interactions require that the software deliver results at the right time, as argued in the guidance document DO-178C (Jacklin, 2012) and in previous Federal Aviation Administration (FAA) reports. Correct timing, in turn, depends on the execution time of individual programs. Hence, the problem of finding an upper bound on the execution time of a program, called Worst-Case Execution Time (WCET) analysis, is an important step in avionics certification.

Unfortunately, WCET analysis is difficult for several reasons. A program can typically execute a large number of different paths. During the execution of one path, the program uses resources in a complex way and this resource use is difficult to analyze. Finally, during the execution of one path, the speed of execution depends on temperature, which, in turn, depends on earlier execution.

As a result, the state-of-the-art and state-of-the-practice in WCET analysis today have limitations in terms of (i) the maximum size of the programs that can be analyzed, (ii) the complexity of the hardware that can be analyzed, and (iii) the ability to analyze hardware that does not have documentation available.

Thus, it is worth exploring ideas to overcome these limitations. The disciplines of machine learning (ML) and artificial intelligence (AI) provide general methods for obtaining knowledge from observations; and for using knowledge for prediction and decision-making; and doing so in uncertain, unknown, or complex environments. These general methods appear to hold promise for dealing with the limitations of current WCET analysis methods.

This report assesses the use of machine learning (ML) and artificial intelligence (AI) to find the WCET of avionics software. This report includes (i) background to the problem and challenges, and (ii) a new method (based on ML) that we have developed for WCET analysis, and (iii) a tool based on this method; we call the method “learn-and-extrapolate.”

1 Introduction

Many parts in aircraft today rely on software that interacts with its physical environment. Typically, this interaction involves taking sensor readings, sending actuation commands, reading commands from humans, and presenting information to humans. These interactions require that the software deliver results at the right time, as argued in the guidance document DO-178C (Jacklin, 2012) and in previous Federal Aviation Administration (FAA) reports. Correct timing, in turn, depends on the execution time of individual programs. Hence, the problem of finding an upper bound on the execution time of a program, called Worst-Case Execution Time (WCET) analysis, is an important step in avionics certification.

Unfortunately, WCET analysis is difficult for several reasons. A program can typically execute a large number of different paths. During the execution of one path, the program uses resources in a complex way and this resource use is difficult to analyze. Finally, during the execution of one path, the speed of execution depends on temperature, which, in turn, depends on earlier execution.

As a result, the state-of-the-art and state-of-the-practice in WCET analysis today have limitations in terms of (i) the maximum size of the programs that can be analyzed, (ii) the complexity of the hardware that can be analyzed, and (iii) the ability to analyze hardware that does not have documentation available.

Thus, it is worth exploring ideas to overcome these limitations. The disciplines of machine learning (ML) and artificial intelligence (AI) provide general methods for obtaining knowledge from observations; and for using knowledge for prediction and decision-making; and doing so in uncertain, unknown, or complex environments. These general methods appear to hold promise for dealing with the limitations of current WCET analysis methods.

This report assesses the use of machine learning (ML) and artificial intelligence (AI) to find the WCET of avionics software. This report includes (i) background to the problem and challenges, and (ii) a new method (based on ML) that we have developed for WCET analysis, and (iii) a tool based on this method.

1.1 Problem and challenges

Traditional research in WCET analysis has focused on the challenges that the input space of a target program can be very large, the state space of a target program can be very large, and the hardware (which influences the execution time) can be very complex. Our research in WCET is driven by the observation that hardware vendors today typically do not disclose details about the

hardware. Hence, our research is driven by the challenge of performing WCET analysis with undocumented hardware.

1.2 New method

The proposed approach is based on an idea that relies on learning and extrapolation. Specifically, the idea is as follows:

1. Run the target program (i.e., the program whose WCET we seek) many times with randomly-generated inputs and measure the execution time.
2. Learn a model that predicts the execution time of the target program given an input.
3. Solve the optimization problem that finds the input that maximizes the predicted execution time. This provides an input for which one can guess that the execution time of the target program will be large.
4. Run the target program with this input and measure the execution time. This measured execution time provides a WCET estimate.

1.3 Tool

The tool has the advantage of being versatile and convenient:

- it does not depend on documentation of hardware
- it does not depend on the target program being written in any specific programming language
- it does not depend on the source code of the target program being available
- it does not depend on any specific hardware features for tracing or monitoring
- it does not change the timing of the target program (i.e., no probing effect)

1.4 Experimental platform and setup

Several experiments were conducted using compiled C code running on Linux on an x86 multicore processor. When training, we used almost all processor cores. However, when running the target program, we used only one processor core (all other processor cores were idle) and the target program was not allowed to migrate between different processor cores (setting affinity with a POSIX call) and it was not allowed to be preempted (setting it to highest priority with a POSIX call). To decrease execution time variations, we did the following:

1. disable hyperthreading,
2. set processor clock frequency to lower value, and
3. disabling Linux CPU throttling.

The computer was connected to a network and it ran a graphical user interface. We did not move the mouse cursor or hit any keys. It is likely that the computer sent and/or received Ethernet frames and this can interrupt the execution of the target program (this was done intentionally because the proposed method was designed to deal with these disturbances will be explained later in this report). Early in the project, a computer with an Intel processor was used but later switched to a computer with an AMD processor (because the latter had more memory—256GB—that we needed). Therefore, comparing results in different tables between different sections are not necessarily consistent. For example, our bubblesort program has average execution time of approximately 0.0036 seconds on Intel CPU but it has average execution time of approximately 0.0023 seconds on AMD CPU.

1.5 Convention: If unit of time is not specified, it is assumed seconds. Results

The key findings are as follows:

1. Our new WCET analysis method performs well for some target programs in the sense that it can find an input that causes long execution time. This is caused by counter-intuitive effects that normal WCET analysis tools do not consider.
2. There are target programs for which our new WCET analysis method does not perform well.
3. Hence, our WCET analysis method should be viewed as a complement to other WCET analysis methods.
4. For our WCET analysis method, how to measure execution time is critical. We have developed a method so that we can tolerate noise.

For our WCET analysis method, one must choose a family of functions that perform prediction of input to target program to a predicted execution time. However, choosing an appropriate family of functions is a challenging task. The family needs to be very expressive so that it can describe the reality of the target programs. However, this expressiveness brings a large number of weights that need to be trained, which requires a large amount of data. We have explored this for affine functions and with affine functions that are augmented with other functions that can

model interaction effects between bits in the input to the target program. More research is needed on how to choose a function family and perform training that suits the special constraints required in our WCET analysis method (ability to extrapolate, having low max magnitude of error) that are different from typical machine learning.

2 Background

2.1 The role of WCET analysis in ertification

2.1.1 About DO-178C

DO-178C is a document that provides guidance for certification. According to DO-178C (2011, p. 67), certification applies to aircraft, engines, or propellers. As part of certification, parts are approved, and this includes the approval of software. DO-178C is about the approval of software as part of certification. An applicant (e.g., organization making an aircraft) seeks certification from a certification authority (e.g., FAA).

DO-178C (2011) specifies objectives, activities, and evidence. DO-178C involves an applicant submitting documents with plans and documents with accomplishments related to the plans. It specifies the type of information that should be in these documents, but it does not mandate the use of any specific requirements such as elicitation process, software architecture, or programming language. In addition, DO-178C does not mandate the use of any specific formal verification procedure, testing method, or code review technique. However, it does mandate certain coverage criteria for testing. DO-178C specifies "what" rather than "how," and it tends to be written at a high level of abstraction. Some parts of DO-178C are more detailed: (i) it specifies known pitfalls and requests that an applicant explains how to deal with them, and (ii) it conveys some experience/wisdom.

2.1.2 DO-178C and real-time requirements

DO-178C (2011) emphasizes the importance of the real-time requirements of software. This is stated explicitly in Section 2.2.2, Section 2.2.3, Section 6.3.1c, Section 6.3.2c, Section 6.4.2.2e, Section 11.10e, and Section 12.1.1d. It is also referred to implicitly in Section 4.4.3, which points out that an emulator may have a different speed than real hardware and, hence if the real-time requirements of software are satisfied in an emulator, then they are not necessarily satisfied when executing on real hardware. A similar point is made in Section 6.2a, which mentions (in the context of verification) "if the code tested is not identical to the airborne software, those differences should be specified and justified." Yet a similar point is made in Section 11.3d and

Section 6.4.1 discusses performing tests in the target environment, since some errors are only detected in this environment.

2.1.3 DO-178C and WCET analysis

DO-178C emphasizes the importance of the WCET of a program. This is stated explicitly in Section 6.3, Section 6.3.4f, Section 6.4.3a, Section 11.10e, and Section 11.20i (DO-178C, 2011). In addition, Section 6.4.2.2 mentions the importance of not exceeding frame times.

DO-178C makes some general points that do not mention WCET analysis but have impact on WCET analysis:

1. As mentioned above, DO-178C (2011) points out that an emulator may have a different speed than real hardware. This is important because some WCET analysis methods rely on a simulator. For details, see Wilhelm, et al. (2008), Section 6.8. This method uses a simulator and performs symbolic execution so that whenever the execution encounters a situation where two or more successor states are possible (e.g., depending on input to the program, initialization of variables, or initialization of cache), it forks simulations of each of these possibilities. If used in a naïve way, this would produce a combinatorial explosion. However, the method merges paths that are executed. Nonetheless, since the method relies on a simulator, the output WCET may be inaccurate. Indeed, as pointed out in Wilhelm et al. Section 2.3.2 about simulation in general (not specifically the method in Section 6.8): "The results [from a simulator] show large differences in timing compared to the measured values." Regarding this issue of difference in speed between a simulator and real hardware, we imagine two ways of improving the method presented in Wilhelm et al. Section 6.8 in. First, one can use the method presented in Section 6.8 to obtain a WCET estimate and the execution path that leads to the estimate; then compute a test-input that causes the program to execute that path and measure the execution time of the program executing on a real processor (i.e., not a simulator). Second, there may be ways to integrate execution-time measurements within each forking point.

2. DO-178C (2011) Sections 4.4.1 and 12.2.1 indicate that some steps may be performed by a tool, which requires tool qualification so that the confidence that one gets from using a tool is at least as high as what one would get without using a tool. Current practice is for a software practitioner to run the target program and measure the execution time (Wilhelm, et al., 2008, p. 3). Typically, the practitioner exercises some judgement on (i) what is the worst-case input to the program and (ii) when declaring the WCET of the program; the estimated WCET is equal to the measured execution time plus a margin.

Thus, a WCET tool must offer confidence that is at least as high as the confidence offered by this common practice.

3. DO-178C (2011) Sections 6.3.1g and 6.3.2g mention the importance of discontinuities. For WCET analysis, the execution time of a program may depend on input to the program. If the execution time, as a function of input, has discontinuities, then it may be difficult to learn this function, and this may pose a challenge for measurement-based WCET analysis methods.

4. DO-178C (2011) Section 6.3.3d specifies the software architecture that can be verified and also gives as an example: "there are no unbounded recursive algorithms." A similar point is made in Section 6.3.4c, Section 11.7e, and Section 11.7f. This is an important issue in WCET analysis because many of the existing tools and theories for WCET analysis do not allow recursive programs; an exception is the method in Wilhelm et al., Section 6.8, in (2008), which allows recursive programs.

The field of military avionics has a similar concern, as noted in MIL-HDBK-516C section 15.5.2:

> "SSSE flow and execution is deterministic and all processes meet their deadlines under all conditions. Frame rates are compatible with real time system performance requirements and support execution rate requirements. SPA mechanization including priority task assignments, interrupt structure and overall processing control structure is sufficient for safety critical requirements/function processing." (U.S. Department of Defense, 2014, p. 477)

Kästner and Ferdinand (2012) discuss safety standards and WCET analysis tools. They point out that timing and WCET analysis play an important role in all the safety standards considered: DO-178B/C, IEC-61508, ISO-26262, and EN-50128.

Souyris et al. (2005) argues that WCET analysis is critical for avionics, and it is a big challenge for Airbus.

2.1.4 Assurance case

There is an interest in exploring alternative forms of certification. Specifically, there has been interest in moving away from processes and checklists and towards assurance case. An assurance case is a set of claims organized as a hierarchy. The claim at the top of the hierarchy is the claim that we want to be true and for which we want to make a convincing argument. The claims at the bottom of the hierarchy are claims for which we have direct evidence; for example, a

measurement calculation, or a formal method that supports the claim. The hierarchy should be constructed so that for each claim that is not at the bottom, it holds that one can be convinced of this claim based on its children claims. In an ideal world, one should prove mathematically for each claim that is not at the bottom that it is consequence of its children claims. However, in practice, this is usually not possible, so there is a human judgement that one can be sufficiently confident that a claim is true if its children claims are true.

Typically, the top-level claim is "the system is safe," but for the sake of illustration, Figure 1 presents a very simple assurance case with the top-level claim being "Timing of software in flight control system is safe."

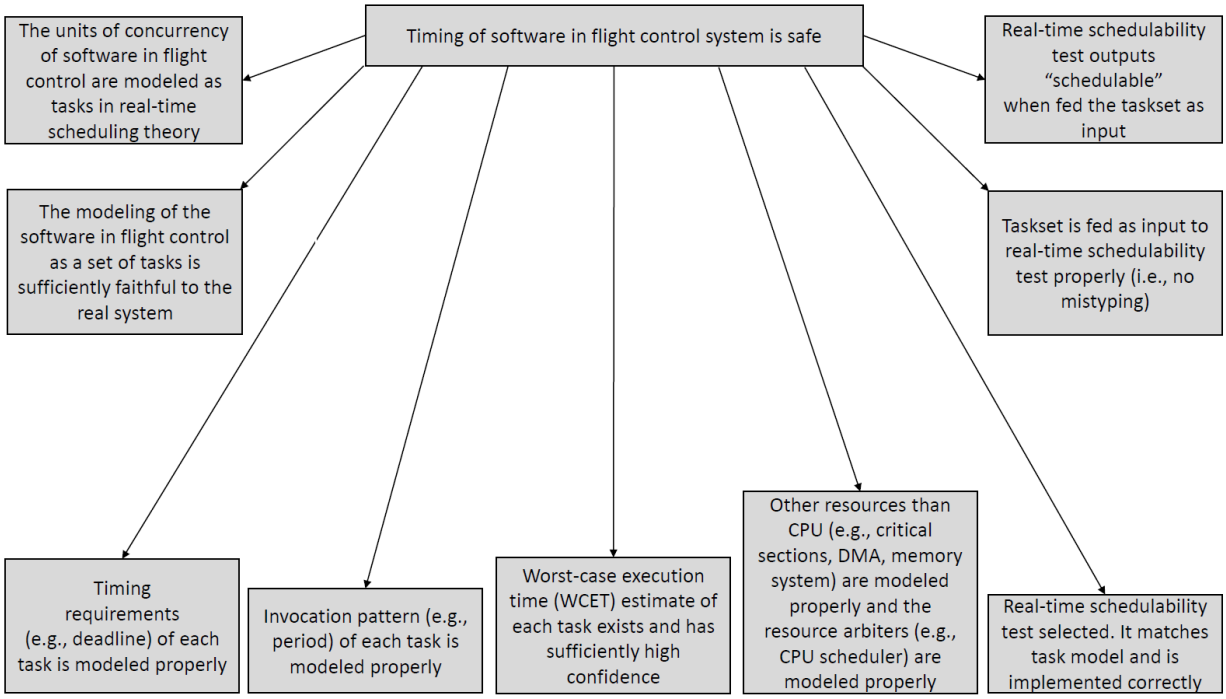

Figure 1. WCET Analysis in assurance case

OA bottom-claim is "Worst-case execution time (WCET) estimate for each task exists and has sufficiently high confidence." Thus, WCET analysis fits naturally in this assurance claim.

Assurance case is currently an area of active research that involves (i) different types of notation (e.g., goal structuring notation) and (ii) ideas on how to get justified confidence in a claim based on its children.

2.1.5 Tool qualification

DO-330 (2011) provides tool qualification guidance. It involves identifying how a tool (e.g., compiler, test-generation, static verification tool) will be used in the software life cycle and

determines the impact on safety of the failure of a tool for each use. For example, the failure of one tool may have strong negative effect on safety (e.g., a compiler generates incorrect code) whereas the failure of another tool may have much less negative effect on safety. Also, note that a tool may be used in different activities where its use in some activities, its failure may have strong negative effect on safety but in other activities, this may not be the case. Based on this determination, for each tool, for each of its use, a tool qualification level (TQL) is assigned. There are five different TQL. For the TQL with a stronger negative effect on safety, the tools and their assigned TQL use require more rigorous tool qualification. DO-330 states the testing coverage criteria to be used for each TQL. For example, for TQL-1, the required level is modified condition/decision coverage. If a tool uses external components (e.g., file system), then they need to go through testing as well (p. 27). Note that qualification may be performed on the use of a tool (one of its features) rather than the entire tool.

DO-330 (2011) emphasizes that it is better that a tool produces no output than producing the wrong output. DO-330 also emphasizes that some tools use external components (e.g., file management) and these should be documented. It is noteworthy that many formal verification tools depend on satisfiability problem (SAT)/satisfiability modulo theories (SMT) solvers or mathematical optimization solvers (e.g., mixed-integer linear program).

Some tools that have been found to be defective include the following: SAT/SMT solvers have recently been found defective (Mansur, Christakis, Wüstholz, & Zhang, 2020; Park, Winterer, Zhang, & Su, 2021; Winterer, Zhang, & Su, 2020); these are foundations in model-checking tools. It has also been observed that model-checking tools have defects (Zhang, et al.). Particularly serious defects (which have been observed) are cases where a verified system is unsafe but the tool outputs safe. In addition, theory for real-time schedulability analysis for controller area network has been found defective as well (Davis, Burns, Bril, & Lukkien, 2007).

2.1.6 Explainability

Given that formal methods can have defects, and these can be serious (outputting safe when it is unsafe), it is natural to ask how to deal with this. One idea that we have proposed recently is the notion of explainability of real-time systems and their analysis (Martins, 2022).

Explainability of real-time systems and their analysis is a broad term. However, one aspect of it is the following. Traditional formal verification tools take as input a model and correctness property and outputs an answer. Typically, the answer is binary (YES/NO or SAFE/UNSAFE) that indicates whether this property is true for all executions that are possible according to this model; however, an answer can also be other quantities. For example, one variant of WCET

analysis is: given a model M of a program and the property φ being that for each execution of the program, it holds that the execution time of this execution is at most 42 milliseconds, give an answer whether this property φ holds. Another variant of WCET analysis is: given a model M of a program, output the WCET of the program.

It would be desirable to change these traditional formal verification tools so that in addition to producing an output as an answer, they should also output a human-understandable explanation for how this answer was obtained. For example, for WCET analysis, we would like the tool to output the WCET estimate and an explanation for how this WCET estimate was produced.

The method and tool created in this project has this type of explainability because it produces a WCET estimate and an input to the target program that causes this WCET estimate. This is useful to a software practitioner in two ways. First, a software practitioner can re-run the target program with this input and check if the execution time is approximately equal to the WCET estimate produced by the tool (if it is not approximately equal, then it indicates that there is a defect in the WCET analysis tool, we have not experienced this). By repeatedly doing this, if there is approximate agreement, a software developer can gain confidence in the WCET analysis tool. A certification authority can also gain confidence as well.

2.2 Why WCET analysis is difficult

The execution time of a program depends on two factors: the path that the program executes, and how resources are used during the execution of this path and dependencies between instructions of this path. The WCET of a program also depends on these factors. These factors are discussed below using example programs. The programs are not representative of real-world programs, and not all of them are difficult to analyze, but they were selected to illustrate the factors that make WCET analysis difficult.

2.2.1 Paths

A program typically has many paths; which path is executed depends on the input to the program and on the initial state of program variables. The number of paths is typically so large that it is not possible to enumerate all of them. In addition, it is difficult to know the exact set of paths that a program can take. These issues are discussed below.

2.2.1.1 Many paths

A program tends to have more than one path because a program is typically written with if-statements and while-statements. To understand why the number of paths can be very large,

consider the program below:

```
if (a1>=100) { flag1 = 1   } else { flag1 = 0   }
if (a2>=100) { flag2 = 1   } else { flag2 = 0   }
if (a3>=100) { flag3 = 1   } else { flag3 = 0   }
...
if (an>=100) { flagn = 1   } else { flagn = 0   }
```

Since there are n if-statements, there are 2^n different paths. If n is large (say n=200), then the number of paths is so large that, with current computers, it is not possible to explicitly enumerate all paths within a reasonable amount of time.

2.2.1.2 False paths

Consider the program listed above but change the first condition of the if-statement in the first line to (a2< 100). Then we get the program below:

```
if (a2< 100) { flag1 = 1   } else { flag1 = 0   }
if (a2>=100) { flag2 = 1   } else { flag2 = 0   }
if (a3>=100) { flag3 = 1   } else { flag3 = 0   }

...
if (an>=100) { flagn = 1   } else { flagn = 0   }
```

One can now ask, is there a path in which the first condition is true and the second condition is true? That is, is there a path in which both (a2< 100) and (a2>=100) are true? The answer is "no," and the reason is that there is no a2 such that both of them are satisfied. We say that the path with (a2<100) and (a2>=100) is a false path. In this case, one can see that there are only 2^{n-1} paths in the program (as opposed to 2^n paths in the program above). In this particular program, it was easy to identify a false path. It is also easy to identify the set of false paths and to identify the paths that the program can take. In other programs, it is not easy.

2.2.1.3 Uncertainty about number of loop iterations

Choose a fixed n of program 1 above and embed it within a while loop where the condition of the while loop depends on a variable that is modified in each loop iteration so that the long-term consequences of the modifications are difficult to predict. The following is an example of such a program:

```
while (k>1) {
  if (a1>=100) { flag1 = 1   } else { flag1 = 0   }
  if (a2>=100) { flag2 = 1   } else { flag2 = 0   }
  if (a3>=100) { flag3 = 1   } else { flag3 = 0   }
  ...
  if (an>=100) { flagn = 1   } else { flagn = 0   }
  if (k%2==0)  { k     = k/2 } else { k = 3*k+1   }
}
```

In this program, it is not trivial to prove an upper bound on the number of loop iterations. Even if the program is modified to initialize the variable k to some specific value, then for many values, it holds that it is still non-trivial to prove an upper bound on the number of loop iterations. For example, if k is initialized to 27, then one can show (as shown by the Collatz Conjecture https://en.wikipedia.org/wiki/Collatz_conjecture) that the iterations go through 111 steps, but it is not trivial to "see" that without running the program.

2.2.1.4 Randomness

Some programs rely on randomness:

1. Artificial neural networks (ANNs) are often initialized with random weights. They then undergo training that iteratively changes the weights to minimize a cost function. Typically, the program performing the training terminates when the weights have been set so that an external evaluation (cross validation) declares that the weights yield a sufficiently low cost on another data set. Clearly, in this case, the execution time of the program that performs the training depends on the initialization, which is random.
2. Rapidly exploring random trees (RRTs) is a commonly used motion-planning algorithm for autonomous cars. It randomly generates points in a state space, and then runs a shortest path graph algorithm to connect them to form a trajectory from the initial state to a goal state. Once such a trajectory is found (or a trajectory with sufficiently low cost is found), the program terminates. Once again, the execution time depends on random numbers. Typical implementations of the aforementioned programs use pseudo-random numbers rather than random numbers, but it is not obvious that the use of pseudo-random numbers makes the WCET analysis problem easier than if they were random numbers.

2.2.1.5 Knowing the set of possible inputs is difficult.

1. Explanation: The set of possible paths of a program typically depends on the set of possible inputs to the program. This can be seen in Section 2.2.1.1. If (a1, a2, …, an) are inputs to the program, then the path depends on the input. Thus, the set of possible inputs determines the WCET of the program. Often, the set of possible inputs to a program depends on the physical environment. For example, for an image-processing application taking an image at 1024x768 resolution with 24-bit color, one may think that the set of possible inputs is equal to the set of possible images that can be created that have 1024x768 resolution with 24 bits. This would be $2^{1024*768*24}$ possible images and hence $2^{1024*768*24}$ possible inputs to the program. However, the image-processing application may be installed in a physical environment where only a subset of these images is possible. Unfortunately, often one does not know the set of possible inputs to the program. One can, however, find and use an over-approximation of the set of possible inputs to determine WCET.

2. Example: An industrial example of this is from the research literature in Section 4.1 in Puschner & Koza (1989), which describes an image-processing application in manufacturing. The program takes an image and iterates over all pixels in the image, where each pixel of a special color causes a longer execution time. The authors point out that because one has knowledge of the physical environment (the position of a camera and the possible positions of objects on a conveyor belt), one can use this knowledge to find a useful over-approximation of the set of images. The authors note that in this application, one knows that, for each image, the number of pixels with the special color in this image ≤ THRESHOLD. The over-approximation, then, is the set of images where the number of pixels with the special color in this image ≤ THRESHOLD

2.2.1.6 Pointers

A pointer is a variable that stores a value that is used to refer to a function or data. Typically, a pointer is an integer that represents a memory address. The path of a program may depend on a pointer, as shown in the program below:

```
int temp = 5
int min(int t1, int t2) {
  if (t1<t2) { return t1 } else { return t2 }
}
int max(int t1, int t2) {
```

```
  if (t1>t2) { return t1 } else { return t2 }
}
int main(){
   q = &temp
   p = min
   p(2,3)
   p = max
   p(*q,14)
}
```

In this program, *p* and *q* are pointers; *p* is a pointer to a function and *q* is a pointer to an integer. When the program starts, the pointer *q* is assigned to refer to the variable temp. In this program, the fourth and the second last line make a function call where the function is determined by a pointer.. In the second-last line, the first argument (*q) to the function call is also given by a pointer, and this determines the path executed in the max function. In this particular program, it is easy to see how the value of the pointers influences the path executed, and it is easy to determine the set of paths that are possible. In many other programs, however, it is difficult to determine the value of a pointer, and it is thus difficult to determine the set of paths of a program.

2.2.2 Resources and dependencies

The execution of an instruction requires resources, and the execution of an instruction may depend on data produced by other instructions. For these reasons, the time it takes to execute an instruction *I* may depend on (i) other instructions that use the same resource as *I* uses (not necessarily at the same time) and (ii) other instructions that produce data that *I* uses. These issues are discussed below.

Processors today typically operate as follows: a processor uses a register called program counter (or instruction pointer) to perform a lookup in main memory; this retrieves a word that is interpreted as an instruction. This instruction is executed on the processor. Then, the next instruction is retrieved from main memory, and this procedure repeats. When executing an instruction, the processor may read or write to memory elements within the processor (called registers), and it may read from or write to main memory.

For most instructions, there is a single fixed number (e.g., 4) such that it holds that after the instruction has executed; the program counter is incremented by this fixed number. There are

other instructions, however, that can modify the program counter in more flexible ways; these instructions are called branch instructions. Some branch instructions change the program counter to a fixed value that is associated with the branch instruction; that is, two different branch instructions may change the program counter in different ways. Other branch instructions change the program counter to a value that is computed based on the value of the content of the registers. Yet another variation of branch instruction allows the program counter to be changed only if a certain condition (computed based on the contents of registers) is evaluated to true; these instructions are called conditional branches.

Many techniques within a processor are used to speed up the execution of a single program. The following list describes these techniques:

1. The execution of one instruction may start although its preceding instruction has not yet finished (pipelining).
2. The processor can execute more than one instruction in parallel (superscalar processor).
3. When the processor executes more than one instruction in parallel, they may finish in a different order than they started with (out-of-order processor).
4. Frequently accessed data items are stored in a small but fast memory called cache memory (or cache for short).
5. Main memory is designed so that certain memory access patterns can be served faster. For example, if two consecutive memory accesses go to the same row in a memory bank, then the first memory access will load the row into a row buffer internally in the memory bank; this allows the second memory access to be served faster.
6. A memory controller queues up memory accesses and reorders the memory accesses before serving them. The reason for performing this reordering is to minimize the time it takes for all memory accesses to finish. This is done considering the overhead mentioned in technique 5 and the following overhead considerations:
    a. Main memory has limitations on the rate of accesses that are allowed. For example, the time from when one memory bank serves a memory access until it serves another memory access must be above a certain bound. There are similar bounds on the allowed timing of service for four consecutive accesses on a memory bank.

b. A memory access must use a memory bus; this bus is, at each instant, set for one direction (read or write). If the direction needs to change, then it takes extra time to serve the memory access.
c. If a write to memory takes a long time, then the program can continue executing; but if a read from memory takes a long time, then the processor may stall because the next instruction may need the data produced by the former instruction.

7. The processor fetches data or instructions to the cache speculatively (i.e., we do not yet know if the program will access this data or instruction).

8. The program starts executing an instruction in the pipeline speculatively (i.e., we do not yet know if the program will need to execute this instruction).

9. The instructions used within the processor are not the same as the instructions of the program. The reason for this is that some instruction sets (e.g., x86) have variable-length instructions for which it is difficult to design a pipeline. Hence, an instruction in the program is translated into other instructions that are actually executed on the processor. Typically, there are internal memories in the processor that store this translation for a block of instructions in the program.

10. The speed at which a processor executes is determined by its clock frequency. For a short duration, it is possible for a processor to execute at a high clock frequency, but after some time, the clock frequency must be decreased, otherwise the temperature of the processor becomes too high. Thus, the time required for executing a given instruction in a sequence of instructions depends on the execution of the previous instructions (because they can heat up the processor), and it depends on the processor temperature when the sequence of instructions began. In addition, some parts of a processor are shut down (in order to save power); waking up those parts takes time. For example, in some Intel processors, the vector-processing unit is shut down (Fog, 2015) and it can take 14µs before it can be used at full speed.

Modern processors use all or almost all these techniques. Some of these techniques can be applied with multiple instances. For example, processors today typically do not just have a single cache but instead four levels of cache and a cache may be separated into an instruction cache and a data cache. In a system with multiple caches, the caches can be quite diverse. The caches typically have different sizes, and a cache miss leads to an access to another cache at a higher level. Caches can be organized internally in different ways. Caches may be fabricated with different circuit technologies (for example, level 1, level 2, and level 3 caches may be static

random access memory (SRAM), whereas level 4 may be dynamic random access memory (DRAM)).

What all these techniques have in common is that they use resources within the processor and within the memory system so that an instruction *I* can execute faster. This assumes that some other instruction has set up state properly (which relies on a resource), no other instruction has interfered with that state, and there are no data dependencies that delay the execution of instruction *I*. An example of how one of these techniques influences execution time can be understood by considering caches. Consider instructions A and B. Instruction A loads data x into the cache. Instruction B executes much later and reads x; this instruction will execute faster because x is already in the cache (cache hit). On the other hand, there may be an instruction C that loads data y into the cache and this evicts data x from the cache; if C executes before B but after A, then the execution of B is slower.

Because these techniques introduce resource conflicts and dependencies between instructions, even for a program with a single path, it can be challenging to find the WCET. One example of this is determining if a given block is in cache (for the reason mentioned in the above paragraph, determining if C can evict the cache block fetched by A). Programs with pointers make this analysis even more challenging because it is difficult to know which address is accessed and therefore which resource in the memory system is being used.

In addition, many processors today use virtual memory systems, where the address generated by the program (called virtual address) is translated into another address (called physical address) used to access physical memory. This translation requires additional resources. There is a small memory called the translation lookaside buffer (TLB) that caches some translations. There is also a table (typically called the page table) or tables that store all the translations. Because it is possible that an instruction generates a virtual address for which the TLB currently does not have a translation, there are cases when it is necessary to perform an additional memory access to the page table.

Furthermore, there is state in a processor that does not affect the computed result and does not maintain a copy of data but instead is state that is used for internal resource management in the processor. For example, Ferdinand, et al. (2001) report that in the ColdFire 5307 processor, there is a counter (called the global replacement counter) that is used to determine which cache block should be evicted when a new cache block needs to be put into a cacheset that is already full. The execution time of a program depends on the initialization of this counter.

In summary, all these techniques affect timing and hence affect WCET. To make things worse, for many resources there is no documentation specifying the behavior of these techniques.

2.2.3 General comments

The challenges of paths, resources, dependencies, and undocumented hardware are well known in the research community. See Wilhelm, el al. (2008) for an example. Practitioners also recognize some of these challenges. Indeed, in the previous FAA study (Federal Aviation Administration, 2005), researchers surveyed software developers in the avionics industry and found that many of them mentioned the difficulties of caches and pipelined processors. Federal Aviation Administration (2006), made similar points.

These aforementioned challenges apply to the case of a single program executing on a single processor. Additional challenges arise for cases that are more general. Preemptive multitasking can cause cache-related-preemption delay (eviction). In multicore processors, one program executing on one processor core can influence the timing of another program executing on another core through sharing of resources in the memory system and through thermal management. In simultaneous multithreaded processors, threads can share CPU resources and hence affect the timing of each other. In addition, in some processor chips, a floating-point unit is shared among processor cores (https://en.wikipedia.org/wiki/Bulldozer_(microarchitecture)). These challenges make WCET analysis difficult. Even ignoring the challenges with preemptive multitasking and multicore processors, WCET analysis is still difficult. For this reason, this report focuses on only a single program executing on a single processor.

2.2.4 Undocumented behavior

As already mentioned, one of the sources of difficulty in WCET analysis is that the WCET depends on the use of hardware resources (within the processor, within the memory system). One needs to know, for each instruction, how long it takes the instruction to execute, and if this time is variable; one needs to know the upper and lower bounds on the time for the instruction to execute and the factors that influence these bounds. In many cases, the knowledge that one has about these factors is lacking or incorrect. This applies not only to hardware but also to other factors. These other factors are discussed in this section.

2.2.4.1 Undocumented hardware

For simple hardware (e.g., 8-bit microcontrollers without cache), it is often possible to obtain per-instruction timing details from documentation (data sheets). It is often not possible to obtain this information for more complex hardware. We call this undocumented hardware. Finding the

WCET of a program executing on undocumented hardware is important in the following situations:

1. The hardware lacks documentation (as mentioned above).
2. The hardware has documentation, but it is not publicly available.
3. The hardware has documentation that is publicly available, but the documentation is incorrect.
4. The hardware has documentation that is publicly available, but one believes the documentation may be incorrect.
5. The hardware has documentation that is publicly available, and one believes the documentation is correct but the hardware is very complex. Since the hardware is very complex, it is laborious to model it and create a WCET analysis for it. Since in many systems, one would like to do a hardware upgrade, typically, one needs to redo the work for each new hardware upgrade. For this reason, there are situations where it is advantageous to use a WCET method that considers the hardware as undocumented, even though one has documentation.

Documentation is important in WCET analysis, as witnessed by the following:

1. A 2008 study considered WCET analysis and remarked, “For modern processors, technical details are often difficult to obtain due to commercial confidentiality.” (Bartlett, Bate, & Kazakov, 2008, p. 3)
2. A 2000 study considered WCET analysis and remarked, “Modeling of processor’s timing behavior has been the way research has been approaching this problem, but this may be impossible in the future since processor manufacturers are not always releasing information about internal structures.” (Lindgren, 2000, p. 1)
3. A 1998 study considered WCET analysis and remarked, “… leading to the fact that the internals of a new CPU core are kept secret.” (Puschner & Nossal, 1998, p. 1)
4. A 2001 study considered WCET analysis and remarked, “Most pipelines are badly documented by the manufacturers. To be of [sic] the safe side, the initial model of the pipeline already reflects pessimistic assumptions about undocumented features.” (Ferdinand, et al., 2001, p. 9)

5. “PBA [Processor Behavior Analysis] is an extremely time-consuming process, with several man-months required to create a reliable timing model of even a simple processor design.” (Seshia & Rakhlin, 2008, p. 3)
6. “However, the features on contention mechanisms are usually very poorly documented.” (Bin, 2014)
7. “Since multi-core COTS are inherently very complex with some undisclosed contention mechanisms and the behavior of applications on shared resources is also a gray- or black-box to users, the variability estimation is very difficult in such circumstance.” (Bin, 2014)
8. “Embedded architectures come with detailed ISA and block diagram, but many aspects of the micro-architecture remain undisclosed such as the exact SoC network topology, contention, arbitration, and prefetcher mechanisms.” (Bin, 2014)
9. “The knowledge of the hardware is usually limited to users.” (Bin, 2014, p. 13)
10. “Additionally the processor description published by the processor vendors is often inaccurate or coarse to hide the intellectual property.” (Bernat, Colin, & Petters, 2002, p. 1)
11. “Thus, the manuals typically have a rather tenuous relation to the chips they are supposed to document, which is a major problem for hardware modeling.” (Engblom, 2001, p. 2)
12. “It should be noted that bugs regarding the timing behavior of chips are quite common while functional bugs are comparatively rare….” (Engblom, 2001, p. 2)
13. “The documentation for a processor should be assumed to contain errors or incomplete information.” (Engblom, 2001, p. 4)
14. “Precisely modeling these features is problematic as on the one side it becomes quite complex and on the other side, exact information of the previous instruction stream cannot be calculated in general. A further problem is that the vendor’s documentation of a processor’s instruction timing is often a very rough approximation of reality.” (Kirner, Wenzel, Rieder, & Puschner, 2005, p. 1)
15. “However, for complex architectures, it is often common industrial practice to experimentally derive the value of WCET. The measurement-based approach may determine a WCET that does not correspond to the absolute WCET. However, for COTS platforms where many micro-architectural details are undisclosed this approach is the

only viable option for practically dimensioning a complex system at design time." (Mancuso, Pellizzoni, Caccamo, Sha, & Yun, 2015, p. 2)

16. "As previously discussed, an exact micro-architectural model is typically not available for COTS systems." (Mancuso, Pellizzoni, Caccamo, Sha, & Yun, 2015, p. 2)
17. "If hardware details are not known, only the ET approach can be applied. If hardware is not available yet but specification of the hardware has been supplied, only SA approach will yield results." (Here, ET means evolutionary testing and SA means static analysis.) (Wegener & Mueller, 2001, p. 23)
18. "Furthermore, it is difficult to verify the correctness of the models that the SA toolset uses, i.e., the approach depends on the availability of detailed and correct data about the cycle level behavior of the actual hardware." (Wegener & Mueller, 2001, p. 23)
19. "… missing or even incorrect documentation (user manuals generally focus on specifying the programming models but do not provide a detailed view of the processor internals nor accurate instruction timings)." (Altmeyer, Lisper, Maiza, Reineke, & Rochange, 2015, p. 4)
20. "… when the processor features complex hardware mechanisms, which are usually poorly understood." (Altmeyer, Lisper, Maiza, Reineke, & Rochange, 2015, p. 4)
21. "The main obstacle remains the availability of the hardware description. Processor manufacturers are very reluctant to provide detailed hardware descriptions out of fear of plagiarism." (Altmeyer, Lisper, Maiza, Reineke, & Rochange, 2015, p. 4)
22. "Unfortunately, sufficiently precise documentation of the logical organization of the memory hierarchy is seldom available publicly." (Abel & Reineke, 2013, p. 1)
23. "Unfortunately, documentation at the level of detail required for sound and precise static WCET analysis is hard to come by. Processor manuals are often ambiguous as they are written in natural language. Sometimes they do not provide any information about a particular architectural feature at all." (Abel & Reineke, 2013, p. 1)
24. "An engineer might contact the processor manufacturer to inquire more details, which the manufacturer is often not willing or able to provide to protect his intellectual property." (Abel & Reineke, 2013, p. 1)

25. “Further the lack of details on processor internals, due to IP restrictions or incomplete specifications, limits the information available for analytic timing models.” (Abella, Padilla, Castillo, & Cazorla, 2017, p. 1)

26. “For instance, in the context of increasingly complex processors, formal proofs cannot be provided on the correctness of timing (cost) models for static timing analysis. Instead, industry resorts to measurements to derive those figures. As an illustrative example, Airbus and AbsInt had to resort to measurements to obtain some latencies for the Freescale P4080 processor [Nowotsch et al. 2014].” (Abella, Padilla, Castillo, & Cazorla, 2017, p. 1)

27. “The presence of intellectual property restrictions and the vastness of hardware documentation increases the risk of considering inaccurate or incomplete timing information [7], ultimately causing STA users to depend on measurement, and reverse engineering for filing the consequent information gaps [93].” (Cazorla, et al., 2019, p. 4)

28. “Most complex pipelines are badly documented by the manufacturers.” (Ferdinand, et al., 2001, p. 9)

29. “A prerequisite is that good models of the processor/System-on-Chip (SoC) architecture can be determined. However, there are modern high performance SoCs which contain unpredictable and/or undocumented components that influence the timing behavior. Analytical results for such processors are unrealistically pessimistic.” (Kästner, Pister, Wegener, & Ferdinand, 2019, p. 1)

30. “However, because a simulator is a hardware model, this method encounters problems associated with producing a cycle-accurate processor model. This is often very difficult due to missing or incorrect information [9].” (Betts, Merriam, & Bernat, 2010, p. 3)

31. “…static analysis is unsafe, as modern architectures are highly complex and thus modelling them is an error prone process, not last [sic] due to lack of documentation,” (Petters, Zadarnowski, & Heiser, 2007, p. 2)

32. “…without those detailed models, it is not possible to derive safe WCET estimates in general for non-trivial hardware architectures.” (Bünte, Zolda, & Kirner, 2011, p. 1)

33. “Further details about the processor cores, such as local cache and scratchpad configurations, are not publicly available, which complicates any analysis which requires this knowledge.” (Griffin, Lesage, Bate, Soboczenski, & Davis, 2017, p. 6)

34. “In particular, if static analysis tools are used to produce a WCET based on a model of the P4080 processor, they would assume greater available cache space than the hardware using this BSP would provide in actuality, resulting in an optimistic WCET that could be catastrophic if used as a basis for system scheduling.” (Bui, Mott, Vance, & Wotell, 2020, p. 14)

Even beyond WCET analysis, the issues of lacking or incorrect documentation have been recognized:

1. A 2016 study (Heule, Schkufza, Sharma, & Aiken, 2016) considered x86-64 architecture and focused on the input/output behavior of an instruction—not its timing. It was found that among 1795 instructions, 50 of them were incorrectly documented.
2. A 2019 study (Dasgupta, Park, Kasampalis, Adve, & Roşu, 2019) also considered x86-64 architecture and studied the in-put/output behavior. The study considered 3155 instructions and found incorrect documentation as well.
3. A 2015 study (Fog, 2015) measured instruction latencies for x86 processors and found discrepancies between the measured latencies and the stated latencies in data sheets.
4. “We review several recent Intel and AMD specifications, showing that all contain serious ambiguities, some are arguably too weak to program above, and some are simply unsound with respect to actual hardware.” (Sewell, Sarkar, Nardelli, & Myreen, 2010, p. 1)
5. “…we introduce the key examples and discuss several vendor specifications, showing that they all leave key questions ambiguous, some give unusably weak guarantees, and some are simply wrong, prohibiting behaviour that actual processors do exhibit.” (Sewell, Sarkar, Nardelli, & Myreen, 2010, p. 1)
6. “Intel have confirmed they add undocumented features to general-release chips for key customers.” (Easdon C. , 2018, p. 3)
7. “2048 undocumented instructions found on one microcontroller.” (Easdon C. , 2019, p. 7)
8. “…the complexity of the hardware mechanisms and policies used, which may result in a system which is very difficult if not impossible to analyse - assuming that the exact details of the hardware are even disclosed!” (Griffin, Lesage, Bate, Soboczenski, & Davis, 2017, p. 2)

9. “We fuzzed that function and found out that it does not give consistent results between Intel and AMD CPUs when SSE2 instructions are used.” (Adrian, 2020, p. Part 4) (Adrian, 2020)

10. About NXP P4080 processor “…discovery of an issue with the potential to cause up to a 50% decrease in local cache usage and substantial degradation in performance.” (Bui, Mott, Vance, & Wotell, 2020)

2.2.4.2 Defective hardware

One widespread defect is known as row hammer (also known as rowhammer), which is a defect in DRAM chips (https://en.wikipedia.org/wiki/Row_hammer). Rowhammer has been studied in the context of computer security, but we believe it can also affect WCET analysis. First, consider a computer that does not suffer from the rowhammer defect; consider the program shown below:

```
flag = 1
while (flag) {
  do_work()
}
```

Assume that the function do_work() does not modify the value of the variable flag. Then this program will not terminate. Now consider the behavior of this program when executing on a computer suffering from the rowhammer defect. With the rowhammer defect, a certain sequence of memory accesses can alter data in a memory location that has not been accessed, assuming these memory accesses go to the same memory bank. Thus, it is possible that although do_work() does not contain any instruction that accesses the variable flag, flag could be modified and the least significant bit of flag could be flipped; then flag becomes zero, and the program will exit the while loop. In this particular case, the rowhammer defect changed a program from being non-terminating to terminating. With similar reasoning, however, one can construct a program so that a path that was a false path for correct hardware becomes a path that can be executed on defective hardware. Clearly, this can change the WCET of a program.

2.2.4.3 Undefined behavior in programming language

Given a system (that includes software and hardware), there is the stated semantics of the system (in data sheets) and there is the actual semantics of the system. The latter may be different from the former because of undocumented hardware. There is also the potential that the stated semantics of the program are different from the actual semantics. This can be an issue when the WCET analysis tool takes, as input, the source code, and the programming language used has

undefined behavior. In particular, the C programming language is commonly used in avionics software (Quora, 2017), and it suffers from undefined behavior. Thus, according to the semantics that the WCET analysis tool assumes, a certain path could be a false path but in actual execution, it can be used. Clearly, this can change the WCET of a program.

2.2.4.4 Incorrect documentation of software

A program often depends on another program or subprogram (e.g., an external solver for mathematical optimization, a library for linear algebra, or a library for data management). In WCET analysis of a target program, one may use information about other programs (i.e., documentation about the other programs). Unfortunately, software documentation is often incomplete and incorrect. An example is mentioned by Tilbrook and McMullen (1990, p. 5): "Many times it is cheaper for us to determine how your system is used through experimentation, than to recover from your outright mis-truths."

2.3 State-of-the-art in WCET analysis

There is an extensive literature on WCET analysis. This section gives an overview of the literature, and discusses its practicality and shortcomings.

2.3.1 The WCET analysis problem

Roughly speaking, WCET analysis involves finding an upper bound on the execution time of a program. More precisely, WCET analysis involves finding a number such that

- for all inputs that the program can experience, in the environment that it is intended to operate in,
- for all memory mappings that the program may run with,
- for all initializations of memory (i.e., values of variables) that the program can experience,
- for all initializations of the hardware (e.g., valid/dirty blocks in a cache memory),

the execution time when running the program in isolation (i.e., no other programs in the system and no interrupt service routines are executed) is at most this number. One is typically interested in finding the smallest such number.

As suggested above, the result of WCET analysis depends on assumptions. Note that the second assumption mentioned above is memory mapping. The reason why memory mapping matters is that the execution time of a program can vary over runs because data is allocated in different heap and stack locations across runs (Vilardell, Serra, Abella, Castillo, & Cazorla, 2019).

For probabilistic WCET analysis, one is typically given a target exceedance probability and one is interested in finding a number such that the probability that the execution time of the program is greater than this number is less than the target exceedance probability.

The best-case execution time (BCET) is defined analogously. One is typically more interested in finding WCET than BCET, but as Wegener points out: “The best-case execution time (BCET) may also be used to predict system utilization or to ensure that minimum sampling intervals are met” (Wegener & Mueller, 2001, p. 2).

2.3.2 Scope

Some researchers have proposed hardware architectures that simplify WCET analysis (Anantaraman, Seth, Patil, Rotenberg, & Mueller, 2003). Other researchers have proposed software designs (how to structure, write, compile, and link a program) that simplify WCET analysis (Puschner & Burns, 2002; Lundqvist & Stenström, A method to improve the estimated worst-case performance of data caches, 1999). Still other researchers have studied WCET analysis of one program when running concurrently with other programs (Bin, 2014). None of these approaches is the primary focus of this report. Rather, this report focuses on a single program running on COTS hardware and assumes only restrictions common to avionics (e.g., we assume the program does not use recursion and does not use pointers).

2.3.3 Overview of WCET analysis

WCET analysis is often categorized as either static or measurement-based. Both approaches output a WCET, but they take different inputs and produce the WCET in different ways. Static WCET analysis takes as input a target program and a model of the processor (including the time instructions needed and the different factors that influence their timing). It uses these to compute the WCET without running the target program. Measurement-based WCET analysis executes the target program (often multiple times). For each execution of the program, it measures properties of the program. These measurements are then combined to produce a WCET.

The dependence of static WCET analysis on a model of the processor is an important limitation. Many processors do not have publicly available documentation of the timing of their instructions) There are often errors in documentation. Even when documentation is available and correct, creating a timing model of the processor based on such a document that is suitable for WCET analysis is laborious and requires specialized skill.

We noted that WCET analysis is often categorized as either static or measurement-based. However, there are two reasons to expand this categorization:

- Some researchers have introduced the concepts of hybrid WCET analysis, which uses ideas from both the static and measurement-based approaches. When using a hybrid approach, one is typically interested in obtaining knowledge about some parts of the program (e.g., execution times of basic blocks) and using that knowledge to compute WCET (often using static WCET analysis techniques). To obtain knowledge about parts of a program, three approaches have been used:
    a) Instrument the program and measure the quantity one is interested in directly.
    b) Instrument the program and measure something else (e.g., the number of times a basic block has been executed) and use these measurements to obtain the quantity one is interested in (e.g., the execution time of a basic block)
    c) Use hardware tracing features, run the program, and let the hardware tracing feature report the quality one is interested in.
- Approaches a) and b) suffer from the probing effect; that is, the program to which measurements apply is not the same as the original program.
- Some researchers have also focused on probabilistic WCET analysis. A common goal in probabilistic WCET analysis is, given a program and a target exceedance probability, find a number such that when executing the program, the probability that the execution time exceeds that number is at most the target exceedance probability

For these reasons, we propose a categorization of WCET analysis based on whether it is static or measurement based and also how measurements are made and whether probabilities are used. The categorization is shown in Table 1 (along with examples of methods that exist in each category).

Table 1. Categorization of WCET analysis

	Static WCET analysis	**Measurement-based WCET analysis**			
			Hybrid WCET analysis		
	Compute a WCET based on a model of hardware.	Measure execution time end-to-end.	Instrument program; run program and measure execution time of basic blocks; compute WCET.	Instrument program; run program and count number of times basic block has been executed; infer execution time of basic blocks; compute WCET.	Use processor with tracing feature; compute WCET based on results of tracing.
May have probe effect	No	No	Yes	Yes	No[1]
Not probabilistic guarantee	(Ferdinand, Martin, Wilhelm, & Alt, 1996), (Lundqvist & Stenström, An integrated path and timing analysis method based on cycle-level symbolic execution, 1999)	(Wegener, Sthamer, Jones, & Eyres, 1997)	(Petters & Färber, Making worst case execution time analysis for hard real-time tasks on state of the art processors possible, 1999)	(Seshia & Rakhlin, 2008), (Lindgren, 2000)	(Kästner, Pister, Wegener, & Ferdinand, 2019)
Probabilistic guarantee	(David & Puaut, 2004)	(Burns & Edgar, 2000), (Hansen, Hissam, & Moreno, 2009)	(Bernat, Colin, & Petters, WCET analysis of probabilistic hard real-time systems, 2002)	-	-

2.3.4 Rationale for hybrid methods with instrumentation or tracing

One can understand the rationale for the hybrid method because many programs have large input space, so generating unguided random test inputs to the program and measuring the execution time end-to-end is unlikely to trigger the input that maximizes the execution time, see the following:

[1] Some hardware tracing functions do alter timing. See, for example, Intel PT. It is claimed to have <5% performance overhead [BranchTrace 2020].

- "However, our perspective on the problem is that current approaches are too black-box in nature and that using knowledge of software and hardware (i.e., application specific knowledge) will improve results." (Bate & Khan, 2011, p. 3)
- Kosmidis, addressing measurement-based timing analysis, states, "Determining a reliable and tight engineering margin is extremely difficult, if at all possible, especially when the system may exhibit discontinuous changes in timing due to unanticipated timing behavior." (Kosmidis, Quiñones, Abella, & Vardanega, 2016, p. 7).

Further, Kirner et al. (2005) argue that end-to-end execution time measurements are inappropriate because of the large number of possible paths in a program.

Another rationale for the hybrid method is based on tracing from the existence of non-intrusive or low-intrusive tracing features in some modern processors. With regard to the processor TriBoard TC179, Zolda & Kirner (2015, p. 65) state, "The processing core includes a special debugging interface, which allows us to non-intrusively capture cycle-accurate timed traces using a Lauterbach PowerTrace device."

2.3.5 Further rationale for measurements

Kirner et al. (2005) argue that the static WCET analysis method has a number of drawbacks:

- It requires annotating the code.
- Difficulties arise when the executable code has a different structure than the source code (due to optimizing compilers), requiring annotations to be made on the executable code.
- It is difficult to model processors accurately.

For these reasons, Kirner et al. (2005) argue that it is worth complementing static WCET analysis with measurements. The authors propose two methods: (1) using genetic algorithms to generate test cases and measure execution times, and (2) using a hybrid of static WCET analysis and measurements. They argue that the Implicit-Path Enumeration approach (used in many static WCET analysis methods) can do a good job of modeling possible paths, but the remaining challenge is to deal with processor state, which argues for using measurements in WCET analysis.

2.3.6 Information required and confidence provided

The type of information needed for a WCET analysis, and which WCET analysis method yields the highest confidence is discussed below. Recall that the WCET depends on the possible inputs. In some applications, the physical environment determines the possible input and hence

determines inputs to the program that cannot happen in the physical world (unrealistic inputs). An example of this is an Engine Management System in which one unrealistic input is high-engine speed and low-injection set point (Maiza, et al., 2017).

Regarding confidence of results, Altmeyer et al. (2015) discussed the sources of doubt on a WCET estimate for both static WCET analysis and measurement-based WCET analysis. They argued that the sources of doubt for static WCET analysis are:

1. they typically rely on annotation to describe the environment of a program (e.g., loop bounds or certain aspects of the physical environment that constrain inputs to the program),
2. they rely on a stated semantics of the programming language or instruction set, but this may be incorrect, and
3. many WCET analyses operate, at least in part, on source code and (because of compiler optimization) there might be no obvious correspondence between parts in the executable code and parts in the source code.

This can happen even if one compiles source code with optimization turned off. Altmeyer et al. also argued that the sources of doubt for measurement-based WCET analysis are (1) path coverage in terms of paths in the program (typically less than 100%), and (2) path coverage in terms of state transition within performance-improving features (e.g., branch prediction) in the hardware (also typically less than 100%).

Other sources of doubt that Altmeyer et al. (2015) do not mention, but that we believe to be important include user-error and mismatching assumptions. A user-error can occur when one program has been annotated (e.g., an annotation stipulating that an integer taken as input will be in a certain range) to specify possible inputs based on knowledge of the physical environment (e.g., a certain engine), but the annotations no longer apply when the program is used in another environment with different possible inputs. Mismatching assumptions can happen when a processor chip is configurable (e.g., in the P4080 processor, the L3 cache can be configured in different ways). Yet another potential mismatching assumption lies in the chip. Some chips require a certain instruction to be executed to make the use of full hardware capability possible. The WCET analysis tool may have assumed that full hardware capability is available, but the deployed chip does not have it (Bui, Mott, Vance, & Wotell, 2020).

2.3.7 Industrial requirements on WCET analysis

Kirner et al (2005) define the requirements for a WCET tool

1. the tool must work with minimal user interaction (specifically, the user should not have to provide infeasible paths),
2. the method must integrate into the development tool chain of customers without modification of tools from the tool chain,
3. the method must be easily adaptable to new releases of software components of the tool chain, and
4. the WCET analysis method must be easy to retarget to different hardware settings.

Bernat et al. (2003) reported a survey made by Reinhard Wilhelm, Jakob Engblom, Stephan Thesing, and David Whalley that aimed to find practitioners' context and their requirements on a WCET tool. The survey findings were

1. timing validation consists of 1-15% of development time, and measuring part of the code constitutes an additional 5-10% of development time,
2. the tolerable learning effort for a WCET tool is 2-10 days,
3. the tolerable analysis time for a WCET is typically 1-120 minutes (but in one case the tolerable analysis time was much larger: 100,000 minutes), and
4. most developers are willing to adhere to coding standards if it helps WCET analysis.

Bernet et al. (2002)also noted that different organizations have different needs. Some organizations (in avionics) are willing to spend large sums of money on tools if it helps safety, but other organizations (outside avionics) are less willing. The former has all source codes, whereas the latter has only some source codes.

Mezzetti and Vardanega (2011) attempted to apply existing WCET analysis tools in a satellite system. They used aiT (https://www.absint.com) to analyze the satellite's software system. Specifically, they analyzed a case in which the system was run on a Leon 2 processor. They found the following limitations (mostly about difficulties with flow facts):

- The software system was large and the computational complexity of the WCET was too high. In addition, the WCET analysis tool failed because the software system had not been annotated. Hence, it was necessary to run the WCET tool multiple times.
- The software system ran multiple modes, but the logic for these modes was scattered throughout different source code files.

- Since the WCET tool requires a human user annotate source code, it is important the user have domain knowledge. This domain knowledge was lost in this case study because the source code was automatically generated from other tools.
- It was hard to obtain flow facts.
- Code generators used dynamic dispatching calls.
- Communications over a software bus caused a loss of type information, which caused imprecision in the WCET analysis.

2.3.8 Adequate or not

WCET analysis has been used successfully in industry. For example, the over-estimation of aiT (a static WCET analysis tool) has been reported to be in the range of 5-25% for use by Airbus—see Souyris et al. (2005). See Section 7 of Kästner, Wegener, & Ferdinand (2012). On the other hand, there are limitations:

- "The availability of hardware models is increasingly defied by novel hardware architectures." (Mezzetti & Vardanega, 2011, p. 3)
- "State-of-the-art WCET analysis approaches are not up to real-life complex industrial software yet." (Mezzetti & Vardanega, 2011, p. 1)

2.3.9 Basic blocks and control flow graphs

A basic block of an executable program is a sequence of instructions that has no branch instruction. Analogously, a basic block of a program source code is a sequence of statements where none of the statements is conditionals (i.e., no if-statement, no while-statement). Thus, a basic block has a single entry and a single exit. A control flow graph (CFG) of a program is a set of vertices and a set of directed edges in which an edge represents a basic block and an edge from one vertex to another represents the condition that if the former basic block has finished execution, then it is possible for the latter basic block to start execution immediately.

In a CFG, a vertex *j* is a successor of vertex *i* if there is an edge from *i* to *j*. A predecessor is defined analogously. Each vertex except the entry vertex has a predecessor. Each vertex except the exit vertex has a successor. Note that some vertices may have two or more successors. For example, if a program has an if-then-else block, then there will be a vertex with two successors (one successor for the case that the condition of the if-statement is true, another successor for the case that the condition of the if-statement is false). As another example, if a program has a while-loop, then there will be a vertex with two successors. One successor for the case that the

condition of the while-loop is true and a new iteration of the while-loop is performed. Another successor is for the case that the condition of the while-statement is false and the the program exits the while loop.

If a program has multiple entry vertices, then one can add a dummy vertex and an edge from the dummy vertex to each of the entry vertices. Then the dummy vertex id labeled as an entry vertex. In this way, we can model a program as having a single entry vertex. Analogously, we can model a program as having a single exit vertex. In the remainder of this document, we will talk about a program as if it has a single entry vertex and a single exit vertex.

2.3.10 Infeasible paths, single-feasible path

When a program executes from start to finish, one can describe the execution as a path throughout the CFG from the entry vertex to the exit vertex. Note, however, that there may exist paths in the CFG from the entry vertex to the exit vertex such that it is impossible for the program to execute this path. Such a path is called an infeasible path. The following program illustrates an infeasible path (Holsti, 2008):

```
if (x<1) {
  compute_for_100_cycles();
} else {
  compute_for_10_cycles();
}
compute_for_30_cycles();
if (x>3) {
  compute_for_200_cycles();
} else {
  compute_for_20_cycles();
}
```

The CFG of the program is shown in Figure 2.

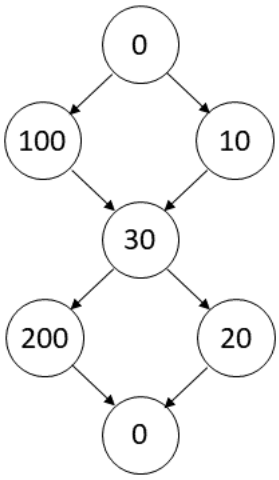

Figure 2. Control flow graph (CFG)

We have labeled the vertices with the time it takes to execute a basic block, and we have assumed (for simplicity) that it takes no time to evaluate a condition in an if-statement. (In a real system, the execution time of a basic block depends on its history; we ignore that aspect in this example to simplify our discussion about infeasible paths.)

Examining the CFG, one can see a path from 0 to 100 to 30 to 200 to 0. It would be tempting to believe that the WCET of the program is 100+30+200; that is, 330. However, for the program to execute this path, the condition of the 1st if-statement must be true and the condition of the 2nd if-statement must also be true. That is, it is required that (x<1) and (x>3). This is impossible. We can conclude that the path from 0 to 100 to 30 to 200 to 0 cannot be executed; hence, it is an infeasible path.

A program can have a single feasible path (SFP), although the CFG may have many paths. See the following example:

```
b = 5;
if (b==5) {
  c = 7;
}
d = 12;
```

Here, we have many basic blocks (because there is an if-statement), but there is just a single SFP.

Ernst has argued that in many embedded software systems, it holds that even if the program has if-statements and while-statements, there is only one path, because the if-statements and while-statements are used for program compaction. The path executed does not depend on input (Ernst

& Ye, 1997). This is SFP as mentioned above. Examples of programs with SFP property are fast Fourier transform (FFT) and finite impulse response (FIR) filter.

3 Previous ideas that use ML/AI for WCET analysis

We give an overview of previous ideas on how to use ML/AI for WCET analysis.

3.1 Genetic algorithms

Wegener et al. explored the use of genetic programming to find the WCET (Wegener, Sthamer, Jones, & Eyres, 1997; Wegener & Grochtmann, 1998). Their idea was that a string specifies the input to a program, and when running the program, one obtains an execution time for this input. The string can be thought of as an individual and the execution time can be thought of as the fitness of that individual. Their approach starts by randomly generating strings (individuals) and obtains execution time (i.e., fitness) for each string. Then new strings (individuals) are generated with the normal operators in genetic algorithms: cross-over (i.e., tails of two strings are swapped) and mutation (i.e., a random bit in a string is flipped). This approach showed good results: It provided inputs to the program for which the execution time is longer than the previously known estimated WCET. The authors developed a similar method to obtain the smallest execution time of the program. They found that, in terms of the algorithm's progress towards finding extremes of execution times, progress towards minimum is faster than progress towards the maximum. This is because there are more solutions with minimum execution time than with maximum execution time, and the execution path for minimum execution time is shorter.

3.2 Genetic algorithms with multi-criteria optimization

Bate and Khan (2011) explored genetic algorithms to find a program input that maximizes the execution time.. Specifically, they ask the question: what should the fitness function of an individual (encoding an input to the program) be? It is natural to use the execution time as fitness (i.e., inputs to the program that causes long execution time correspond to individuals with high fitness). Indeed, other studies, such as that conducted by Puschner and Nossal (1998) have done so. However, Bate and Khan explore the possibility of using a fitness function that depends on more than one characteristic of a run. For example, the fitness of an input could be any of the following:

- a weighted sum of the execution time and the number of branch prediction misses
- a weighted sum of the execution time and the number of data cache misses

- a weighted sum of the execution time and the number of instruction cache misses
- a weighted sum of the execution time and the number of iterations of a given loop

Bate and Khan (2011) explore many different fitness functions based on these characteristics, where the fitness is a weighted sum of two or more characteristics and with different weights. The authors find that there are cases in which having a fitness function based on two or more characteristics makes the WCET analysis better (in the sense of finding inputs that cause a larger execution time).

3.3 Learn details and incorporate these into static WCET analysis

Inductive logic programming is a method in machine learning to learn rules, expressed in logic, from data. Bartlett, et al, (2008) investigates the use of inductive logic programming to determine a bound on the number of times a loop is executed. Inductive logic programming takes as input background knowledge consisting of Horn clauses (for example, a, b and c implies d) that represent facts about the program and sample executions. From this information, input hypotheses are learned. In this case, the hypotheses represent conclusions about loop iterations, which in turn can be used to determine the execution time of loop segments of a program. Due to tractability concerns, there are constraints on how loop bounds can be expressed (technically, loops need to be Presburger loops.)

4 Our research: Measurements

As part of our research, we have conducted experiments on benchmark programs to understand properties of these benchmark programs. Rather than using existing benchmarks, we have written benchmark programs to help us better understand how the programs work.

4.1 Execution-time variation due to smoothness of input changes

Recall that one of the steps in the WCET method that we will develop is learning the execution time of a target program as a function of input to the target program. This learning is based on a set of pairs where in each pair, the first element in the pair is the input to the target program and the second element in the pair is the measured execution time.

Such an idea assumes that we consider a set (potentially infinite set) of functions and we select one of these functions (for example, if the set of functions is represented by weights, then selecting a function can be achieved by assigning values to the weights). This idea tends to be easier to implement if the function is smooth, that is, a small change in input causes a small

change in output. In our context, smoothness means that a small change in the input to the target program causes a small change in measured execution time.

In order to explore smoothness, we do the following:

1. For each targethammingdistance in {1..512} do
    a. do 8192 times
        i. generate a random input
        ii. generate another random input so that its Hamming distance to the former random input is equal to targethammingdistance
        iii. run the target program with these two inputs and measure the execution time
        iv. compute the difference in the execution time between these two runs
        v. compute the magnitude of this difference (e.g., if the difference is -0.001, then the magnitude of the difference is 0.001). For convenience, we call this magnitude of change in execution time when changing input.
    b. compute the average of these magnitude of change in execution time when changing input

By following this procedure, we obtain an estimate of the expected magnitude of change in execution time when changing input by targethammingdistance bits. We say that the function is smooth if it holds that: for small values of targethammingdistance the magnitude of change in execution time is small and for large values of targethammingdistance the magnitude of change in execution time is large.

The results of our exploration shows the function is smooth for the following target programs: bubblesort, matvecmul700, and simplex. The details of the results of our exploration are shown below in Figure 3 through Figure 13 as x-y plots with x-axis being Hamming distance and y-axis being average magnitude of change in execution time.0

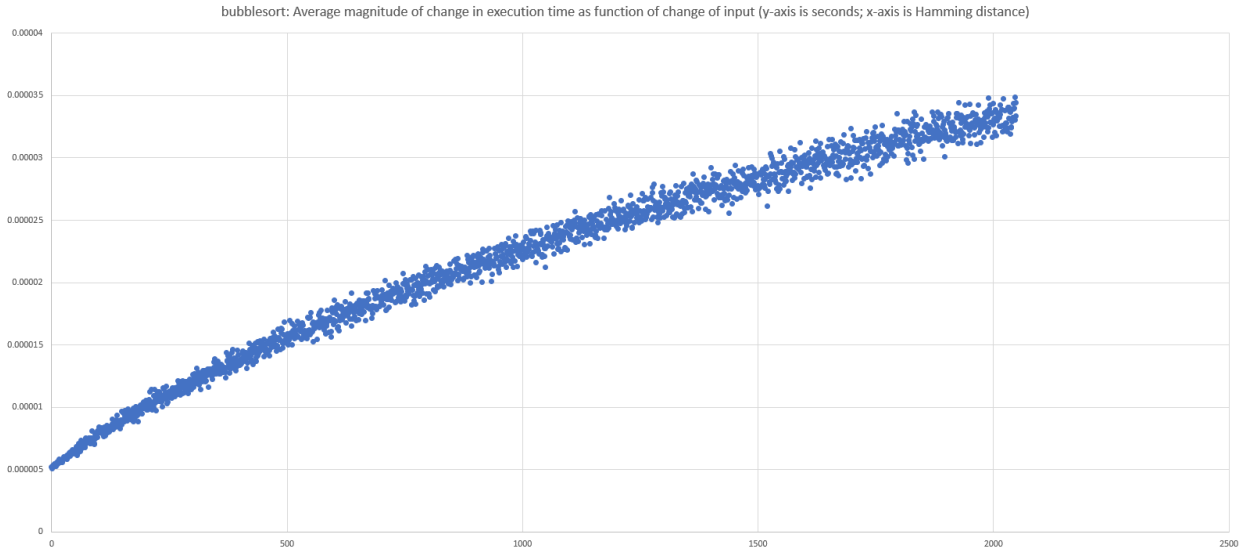

Figure 3. Bubblesort: Average magnitude of change in execution time as function of change in input (Hamming distance)

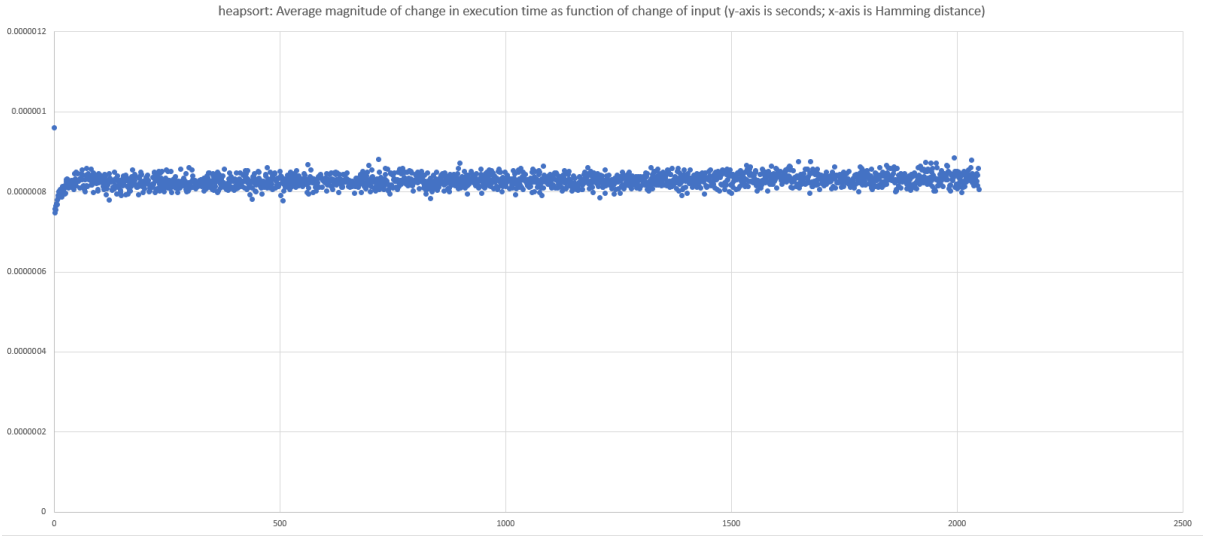

Figure 4. Heapsort: Average magnitude of change in execution time as function of change in input (Hamming distance)

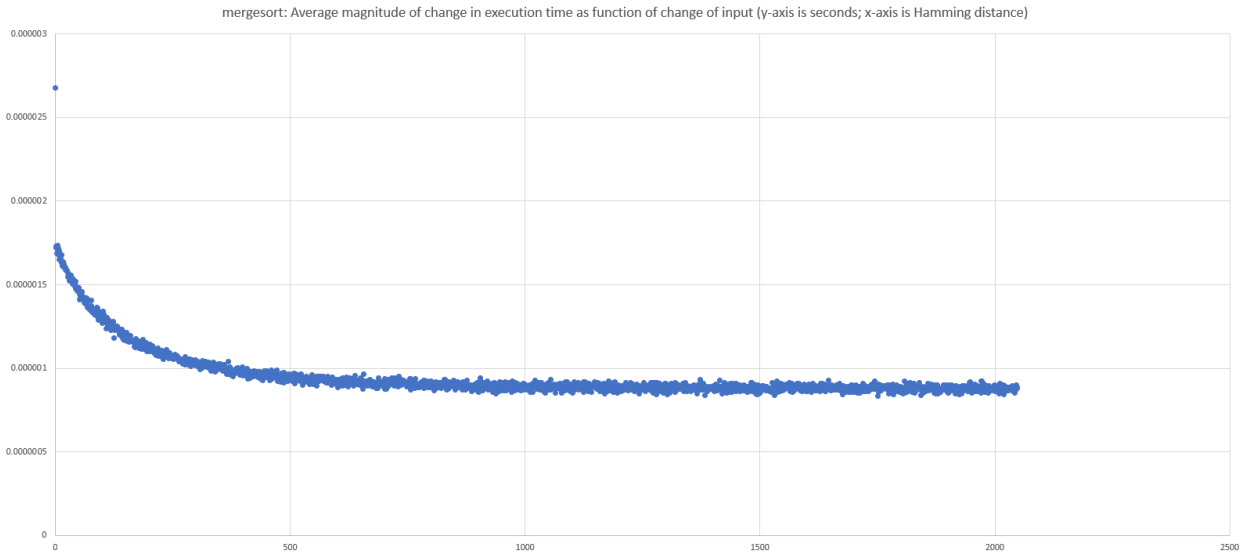

Figure 5. Mergesort: Average magnitude of change in execution time as function of change in input (Hamming distance)

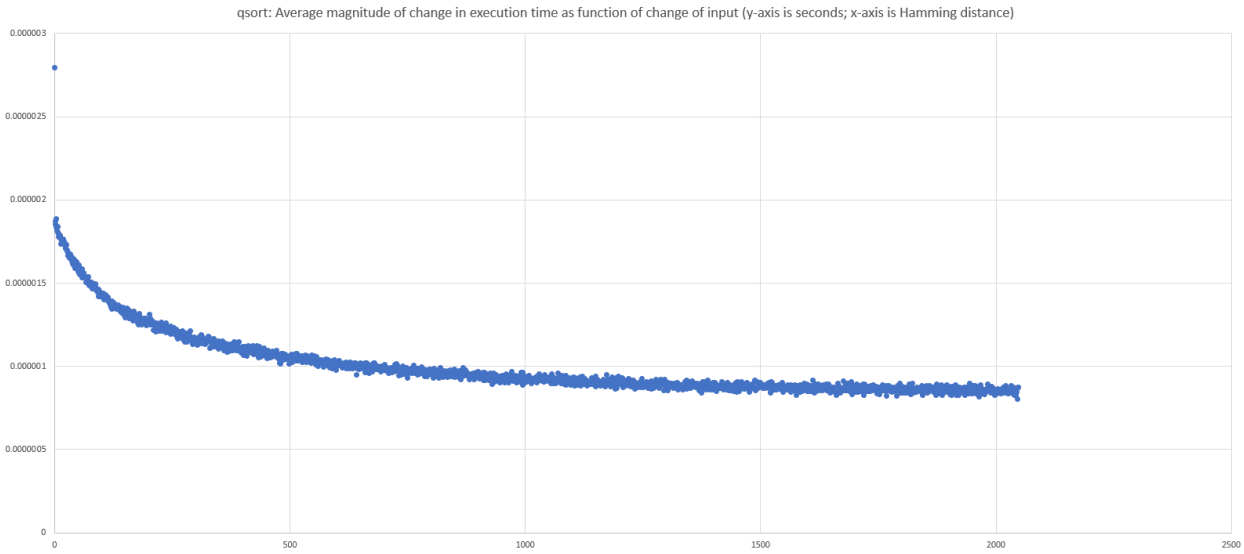

Figure 6. Qsort: Average magnitude of change in execution time as function of change in input (Hamming distance)

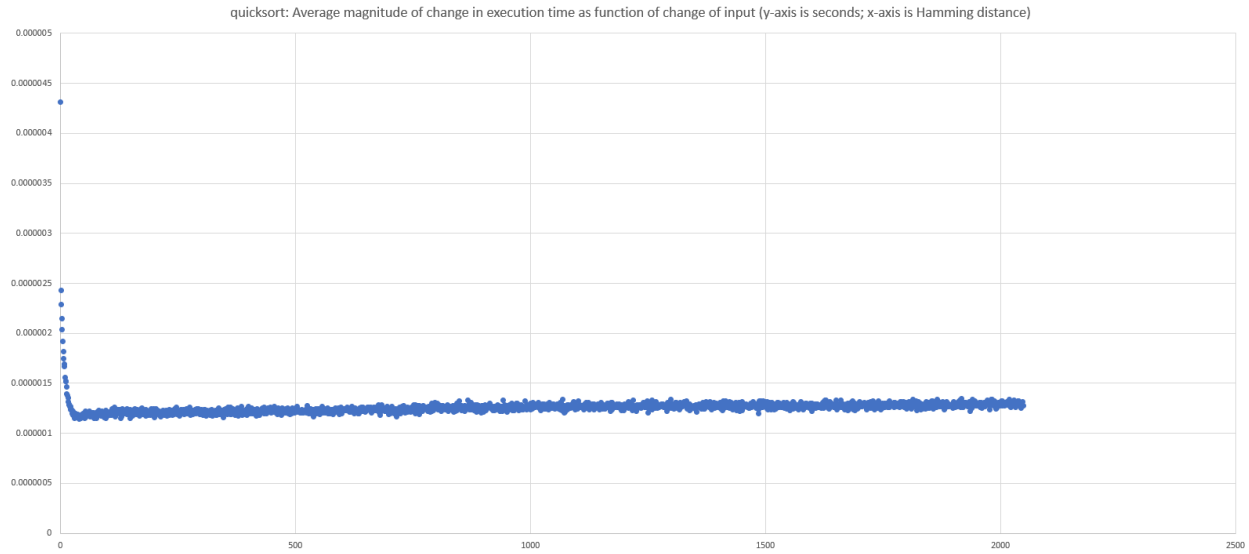

Figure 7. Quicksort: Average magnitude of change in execution time as function of change in input (Hamming distance)

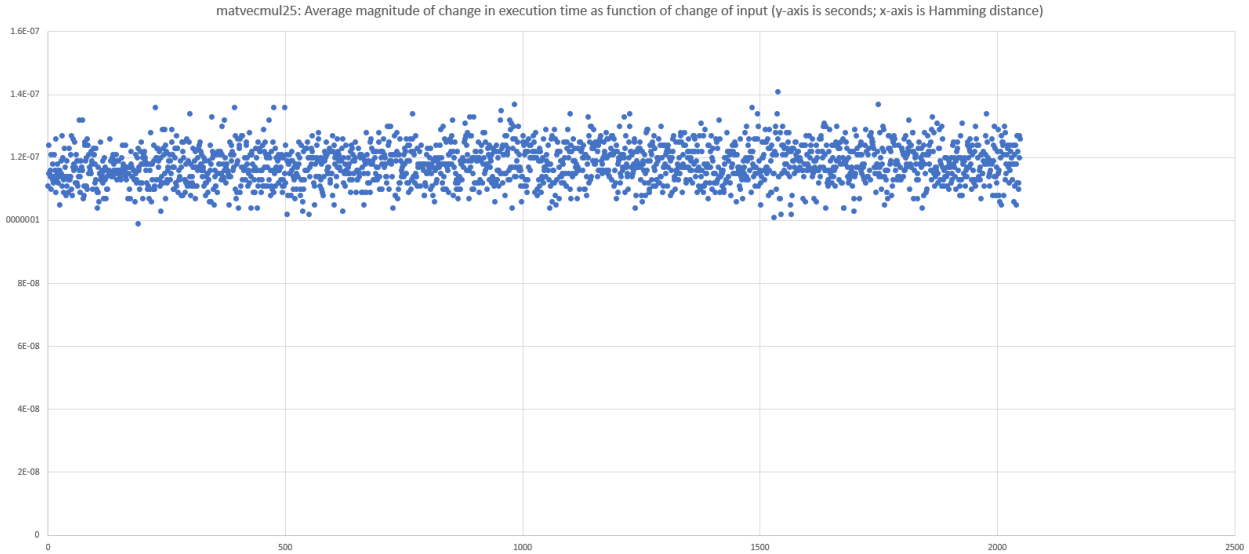

Figure 8. Matvecmul25: Average magnitude of change in execution time as function of change in input (Hamming distance)

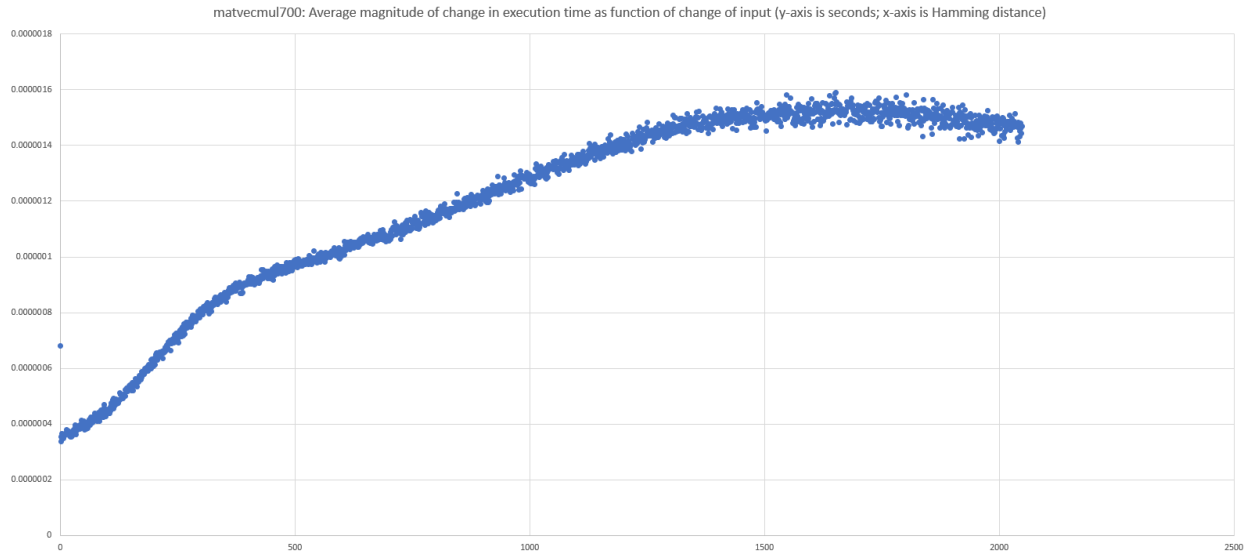

Figure 9. Matvecmul700: Average magnitude of change in execution time as function of change in input (Hamming distance)

.

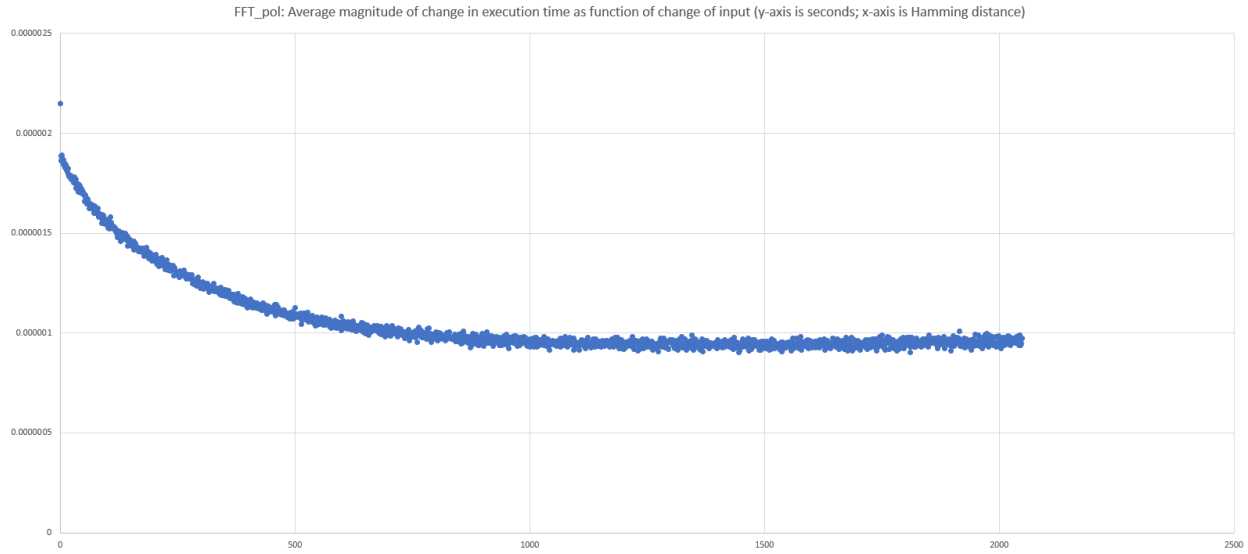

Figure 10. FFT_pol: Average magnitude of change in execution time as function of change in input (Hamming distance)

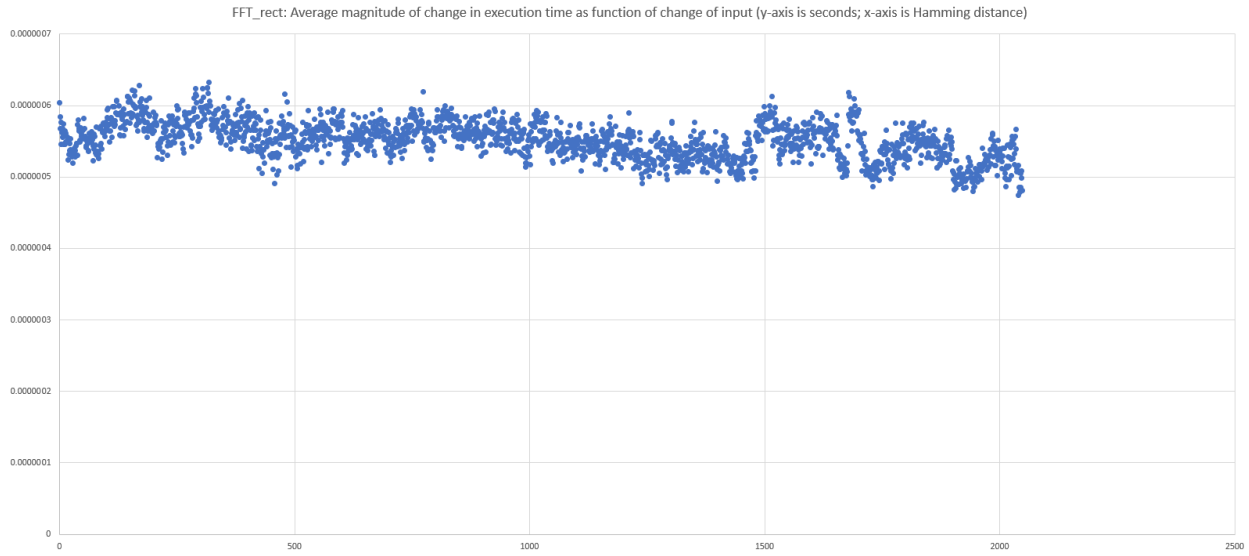

Figure 11. FFT_rect: Average magnitude of change in execution time as function of change in input (Hamming distance)

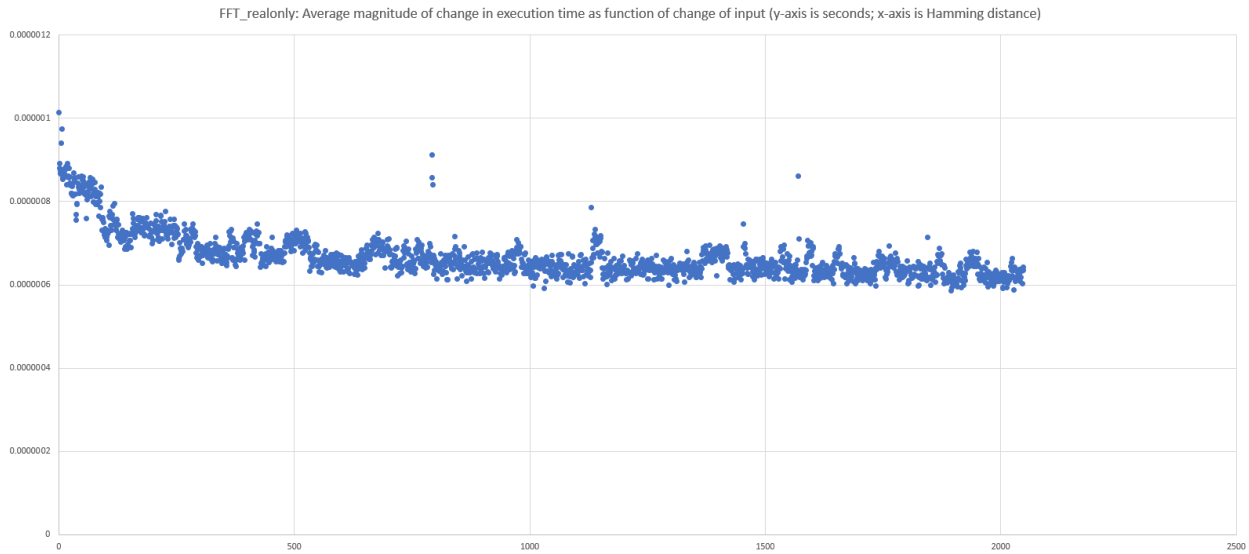

Figure 12. FFT_realonly: Average magnitude of change in execution time as function of change in input (Hamming distance)

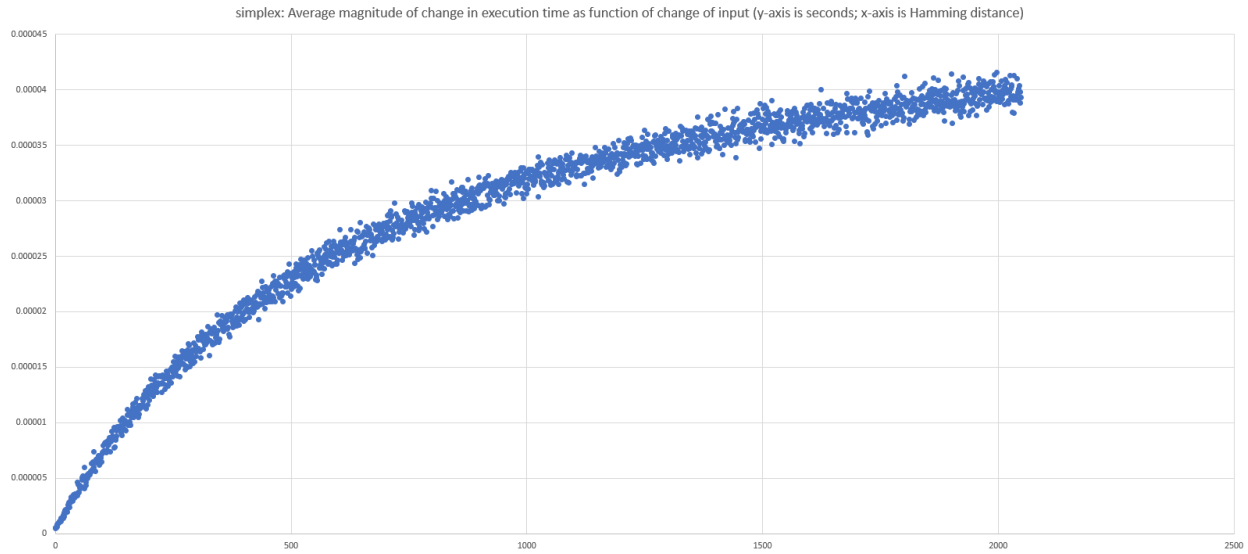

Figure 13. Simplex: Average magnitude of change in execution time as function of change in input (Hamming distance)

Once again, it can be seen from these plots that the function is smooth for the following target programs: bubblesort, matvecmul700, and simplex.

So far, we have measured the distance of two inputs in terms of Hamming distance. This has the advantage that it does not require any interpretation of what the input to the program means. However, by making an interpretation of the input to a program, we can make a more meaningful distance measure. For example, consider a simple program with 8 bits as input. If one input is 01111111 (127) and another input is 10000000 (128), then in terms of Hamming distance, the difference in input is large (in this case 8) but in terms of interpreted value, the distance is small (in this case 1). Therefore, we will now discuss another way to measure smoothness that interprets the bits in the input.

Let x be a one input to the program (a bitvector of N bits) and x' be another input to the program (a bitvector of N bits). Assume that each bitvector represents N/32 32-bit signed integers. Then, we define an error vector e with N/32 elements such that $e = \langle e_0, e_1, \ldots, e_{N/32-1} \rangle$ and with e_k defined as:

$$e_k = as\_int32(x_{32*k:32*k+31}) - as\_int32(x'_{32*k:32*k+31}) \quad (1)$$

We can then define a distance between x and x' as a norm of the error vector e. For example, we might choose the 1-norm; in this case, the distance between x and x' is defined as:

$$\sum_{k=0}^{\frac{N}{32}-1} |e_k| \tag{2}$$

Note that since N can be quite large (N=32768) and since e_k can be quite large as well (e.g., 2^{31}-1), this computed value can be quite large and difficult to interpret. To make it easier to interpret, we divide this by the number of integers. Hence, we define the distance between input *x* and *x'* as:

$$dist(x, x') = \frac{\sum_{k=0}^{\frac{N}{32}-1} |e_k|}{\frac{N}{32}} \tag{3}$$

Writing out the explicit expression for e_k yields that if we choose the 1-norm and divide by the number of elements, the distance between *x* and *x'* is defined as:

$$dist(x, x') = \frac{\sum_{k=0}^{\frac{N}{32}-1} |as\_int32(x_{k*32:k*32+31}) - as\_int32(x'_{k*32:k*32+31})|}{\frac{N}{32}} \tag{4}$$

We call the above "normalized 1-norm distance." We will now conduct an empirical investigation of smoothness using this distance metric. The details of the results of our exploration are shown in Figure 14 through Figure 18 as x-y plots with x-axis being normalized 1-norm distance and y-axis being average magnitude of change in execution time.

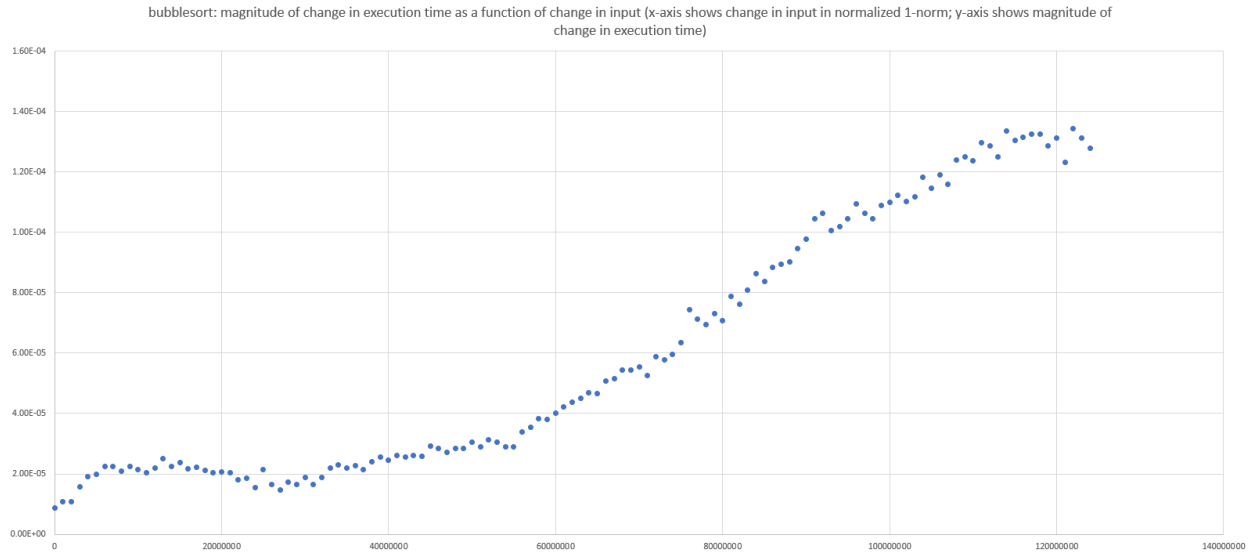

Figure 14. Bubblesort: Average magnitude of change in execution time as function of change in input (normalized 1-norm)

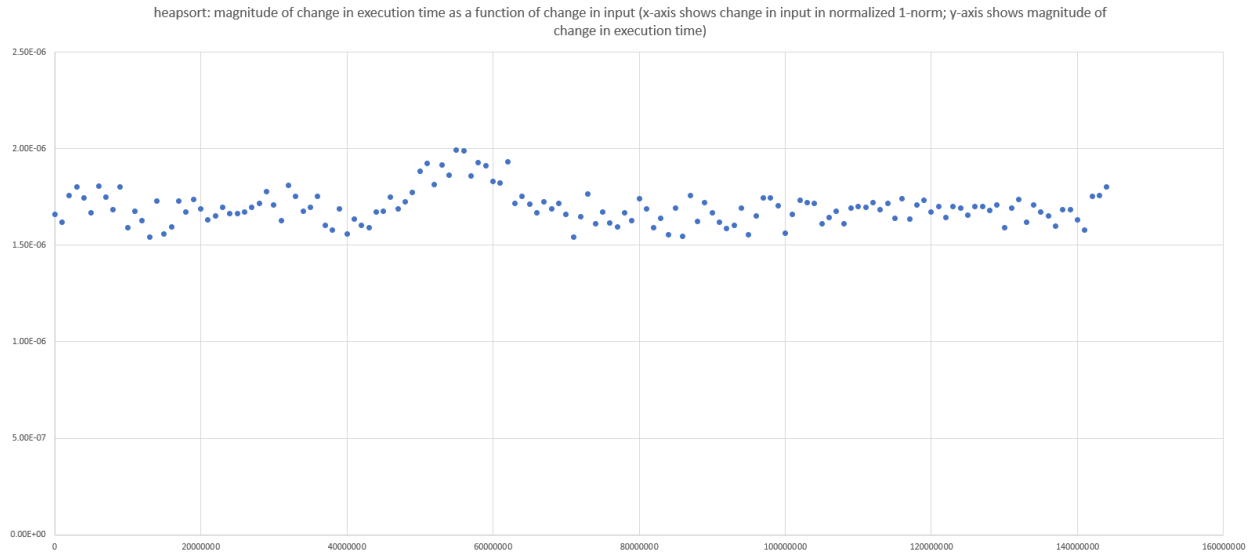

Figure 15. Heapsort: Average magnitude of change in execution time as function of change in input (normalized 1-norm)

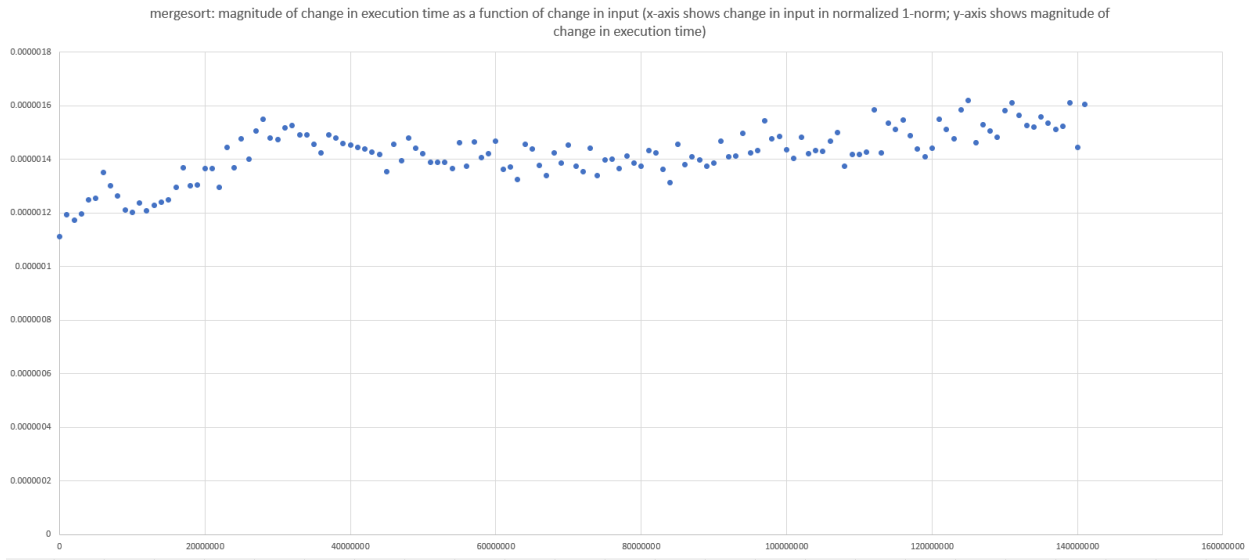

Figure 16. Mergesort: Average magnitude of change in execution time as function of change in input (normalized 1-norm)

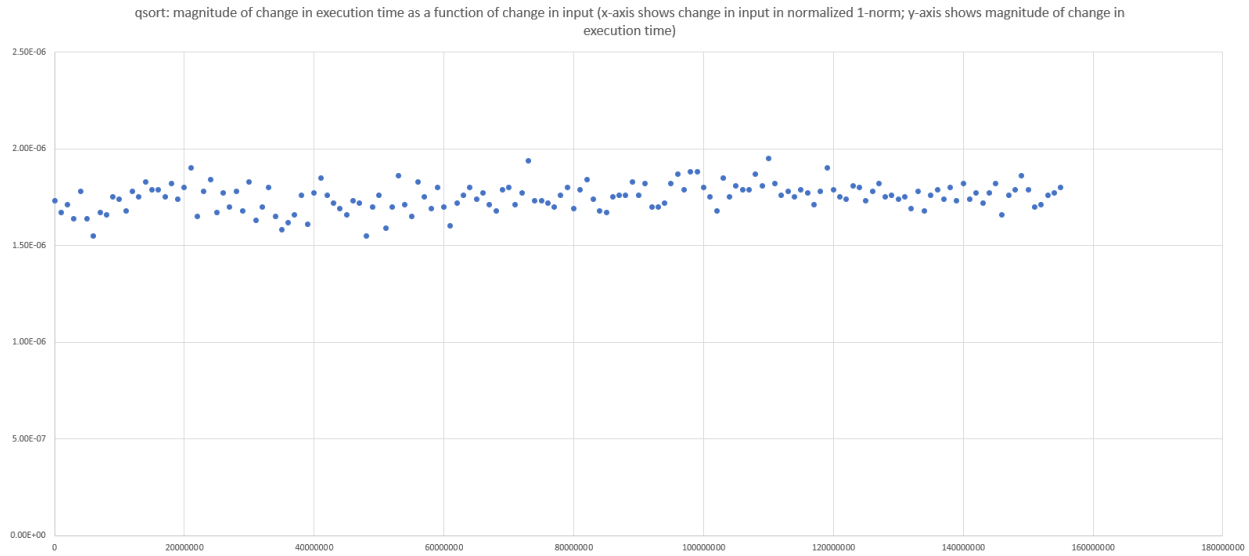

Figure 17. Qsort: Average magnitude of change in execution time as function of change in input (normalized 1-norm)

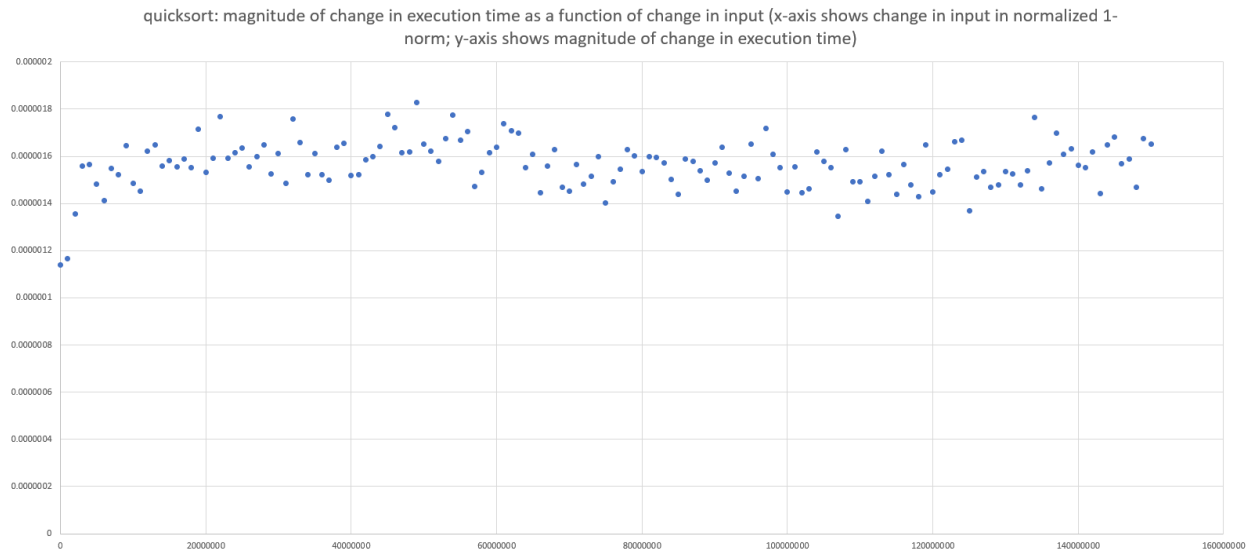

Figure 18. Quicksort: Average magnitude of change in execution time as function of change in input (normalized 1-norm)

In these plots, for bubblesort, it holds that smaller distance causes a smaller expected magnitude of change in execution time but for the other sorting programs, this observation does not seem to hold.

4.2 Non-repeatable timing behavior

The execution time of a target program can be influenced by many different factors. One is the path in the control flow graph that the program exercises. Another is how hardware resources (e.g., cache, branch predictors) are used. The input to the program (and in some programs, the value of global variables when the program starts) influences both. One may then ask, if a target program is run multiple times and for each time, the input is the same and the value of global variables when the program starts is the same, will the execution time be the same?

To answer this, we can first observe that we cannot know the execution time; we can only know the value of the execution-time measurement that we conduct. It may be that we obtain variations simply due to measurement errors. For example, the clock that we use has a resolution of 1/3 nanosecond so it is plausible that measured execution time could vary by 1/3 nano-second for this reason. In addition, even with perfect execution-time measurements, it is plausible that we would observe variations in measured execution time. This variation could be caused by different hardware states that influence timing for different runs. For example, a certain cache block may be in the cache and may be dirty when the program runs once but later this cache block is not in the cache. We have attempted to avoid such variation by running a program twice and only

consider the 2nd run for execution-time measurements. It is interesting to ask, given these considerations, what variations in measured execution time remain?

The following experiment explores this question. Select an input randomly. Then, run the target program 1000000 times with this selected input (i.e., all these 1000000 runs use the same input). For each run, measure the execution time and record it. Compute the sample average of these 1000000 measured execution times. Then, subtract the sample average from these 1000000 runs. This yields 1000000 values that measure the deviation in execution time from the sample average.

Figure 19 shows the outcome of this experiment when conducted for the program bubblesort

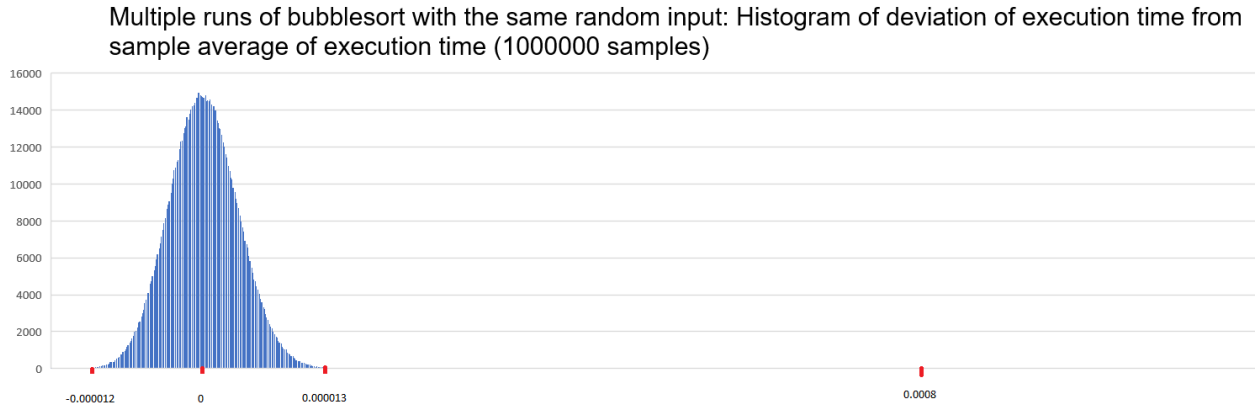

Figure 19. Multiple runs of bubblesort with the same random input.

It can be seen that the deviation in execution time from the sample average appears to be a Gaussian, and in most cases, it is only a few microseconds. With only a few exceptions, the deviation is less than 15 microseconds. By scrutinizing the data further, however, one can see that there are a very small number of runs with a deviation of 0.8 milliseconds (this happens for a very small number of runs so it is cannot be seen in the figure above).

In our program considered, bubblesort, the sample average execution time is 3.7 milliseconds so a deviation of 0.8 milliseconds is large—we did not expect to see this large deviation in particular considering the care we have taken to avoid sources of interference.

4.2.1 Understanding the deviation

We mentioned that we have observed a rare large deviation in execution time when there is no change in input. Now, we attempt to understand it.

After repeating the experiment in the previous section, a clear pattern in when the rare large deviation in execution time was not seen. The large deviation did not necessarily happen in the beginning of the 1000000 runs. It did not necessarily happen in the end of the 1000000 runs. It did not necessarily happen in any specific position (like the 372412th run) among the 1000000 runs, and in many repetitions of the experiments in the previous section, we did not see any large deviation. We have done a large number of ad-hoc experiments to explore various different configurations (in the operating systems, BIOS) to see if we could find a configuration that causes this large deviation to disappear but we did not find any such configuration. As an example, we set the processor clock speed and memory clock speed lower (approximately 1/3 of normal) and this did not cause the large deviation to disappear.

For this reason, we decided to conduct a more systematic exploration; the results of this exploration are presented in this section.

We decided to perform 72000000 runs organized as 72 different batches with 1000000 runs in each batch. For a batch with 1000000 runs, all runs use the same input. However, we allow for the possibility that different batches use different inputs. For each batch, we record the maximum deviation in execution time and note whether the maximum deviation is large (greater than 0.8 milliseconds). We wanted to explore the following questions:

1. Does the existence of a large deviation in a batch depend on time of day?
2. Does the existence of a large deviation in a batch depend on the input? In particular, does it depend on whether the input is randomly generated—which could stress branch-predictors—or whether the input is ordered (ascending or descending)—which may not stress branch predictors?
3. Does the existence of a large deviation in a batch depend on the previous batch?
4. Are there certain events (e.g., interrupts) that are correlated with those batches in which a large deviation exists?

Therefore, we decided to organize these 72 batches into nine blocks with each block having eight batches such that all batches in the same block use the same input to the program. For some blocks, we use random inputs (but with all batches within the block having the same input). For some blocks, we use an array with integers sorted in descending order. For some blocks, we use an array with integers sorted in ascending order. We also record events during each batch (using the Linux /proc file system). There is a large number of possible events in the /proc file system; we recorded all of them and stored all of them. For each batch, we only recorded the number of

events before the batch started and after the batch started (i.e., no recording of the number of events in the middle of the batch); hence this (reading from /proc file system) does not introduce noise in the execution-time measurements.

We use the bubblesort program and conduct such experiments. When reporting results, we let 'R' indicate random input, we let 'A' indicate that the input is an array with integers sorted in ascending order, and we let 'D' indicate that the input is an array with integers sorted in descending order. When reporting time of day, in the interest of brevity, we only state the day of the month and the hour and minute. Also, in the interest of brevity, when reporting results from the /proc file system, we only present the number of function call interrupts because we did not see any correlation for the other types of events. Data related to large deviations are written in color red (to highlight that this is an important piece of data).

Given these goals, questions, and reasoning, we present the results in Table 2.

Table 2. Experiment to detect large deviations in execution time

Batch id	Generate input	Maximum observed deviation in execution time (seconds)	Experiment started at	Experiment finished at	Position of large deviations	Number of function call interrupts
0	R	0.000023445	12; 19:54	12; 21:59		0
1	R	0.000023457				50
2	R	0.000029056				0
3	R	0.000025821				0
4	R	0.000866077	13; 05:03	13; 07:06	0.000866077 at 117627	213
5	R	0.000025268				0
6	R	0.000026581				14
7	R	0.000028032				0
8	A	0.000018722				0
9	A	0.000031576				0
10	A	0.000039367				0
11	A	0.000017841				0
12	A	0.000015069				0
13	A	0.000036907				0
14	A	0.000016855				0
15	A	0.000036792				0
16	A	0.000015151				14

Batch id	Generate input	Maximum observed deviation in execution time (seconds)	Experiment started at	Experiment finished at	Position of large deviations	Number of function call interrupts
17	D	0.000011859				34
18	D	0.000010754				0
19	D	0.000017565				0
20	D	0.000014491				0
21	D	0.000010865				0
22	D	0.000877440	14; 15:01	14; 17:21	0.000877440 at 937313 0.000824893 at 937316	75
23	D	0.000014577				14
24	R	0.000021641				0
25	R	0.000025749				34
26	R	0.000021993				0
27	R	0.000886674	15; 4:15	15; 6:18	0.000886674 at 232977	74
28	R	0.000027254				14
29	R	0.000021712				0
30	R	0.000024440				0
31	R	0.000023776				0
32	A	0.000018319				0
33	A	0.000015539				0
34	A	0.000042138				0
35	A	0.000018702				0
36	A	0.000036919				0
37	A	0.000017868				0
38	A	0.000022107				0
39	A	0.000014482				0
40	D	0.000018483				34
41	D	0.000859000	16; 00:49	16; 03:09	0.000859000 at 755929	74
42	D	0.000010632				14

Batch id	Generate input	Maximum observed deviation in execution time (seconds)	Experiment started at	Experiment finished at	Position of large deviations	Number of function call interrupts
43	D	0.000014959				14
44	D	0.000011267				0
45	D	0.000011876				0
46	D	0.000011947				0
47	D	0.000012056				0
48	R	0.000864327	16; 22:47	17; 00:53	0.000864327 at 148655	108
49	R	0.000027610				0
50	R	0.000027128				0
51	R	0.000882795	17; 05:42	17; 07:46	0.000882795 at 852724	74
52	R	0.000024470				14
53	R	0.000023517				0
54	R	0.000023984				0
55	R	0.000022666				0
56	A	0.000014603				0
57	A	0.000014667				0
58	A	0.000030112				0
59	A	0.000014436				0
60	A	0.000019837				0
61	A	0.000014422				0
62	A	0.000016448				0
63	A	0.000015127				0
64	D	0.000878234				130
65	D	0.000013730				14
66	D	0.000024179				0
67	D	0.000010613				0
68	D	0.000011719				0
69	D	0.000869895	18; 14:53	18; 17:13	0.000869895 at 422518	74
70	D	0.000010217				0

Batch id	Generate input	Maximum observed deviation in execution time (seconds)	Experiment started at	Experiment finished at	Position of large deviations	Number of function call interrupts
71	D	0.000027622				74

It can be seen from the results that a large deviation tends to occur in the batches where the number of function call interrupts is large.

4.2.2 Method to eliminate deviation

Given the existence of rare large deviation in execution time as seen in the previous section, it is natural to ask how to eliminate it. We initially explored ways to disable interrupts on the processor core where we run the target program. This did not eliminate the deviation. We also explored a measurement approach where runs with large number of function call interrupts are eliminated. This worked initially, but as time passed, we found that it did not work anymore. Therefore, the following approach was used:

1. Let REPL be a positive integer (e.g., REPL=5),
2. Run the target program REPL times for each input and measure execution time for each of the REPL runs,
3. For each input, compute the median measured execution time among the REPL measured execution time,
4. For each input, we treat the measured execution time as being the median computed in step 3.

4.3 Distribution of execution time

The research literature has provided experimental results on the probability distribution of execution time with random inputs for the case that the target program is a constraint satisfaction solver; these programs have very large variation in execution time. The research literature has also provided results on extreme-value theory; this provides results on the probability distribution of the maximum among a set of execution time rather than the probability distribution of execution time.

We believe it is worthwhile making our own measurements. Figure 20 through Figure 24 show histograms for different sorting algorithms based on our own measurements. The horizontal axis (x-axis) represents execution time and the vertical axis (y-axis) represents frequency (i.e., how many times did we observe execution time in a certain range).

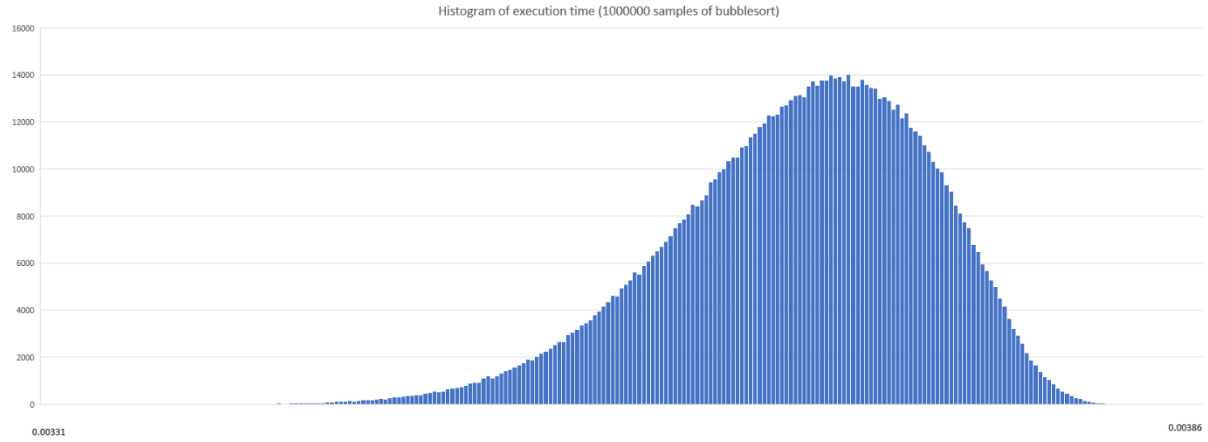

Figure 20. Histogram of execution time using Bubblesort

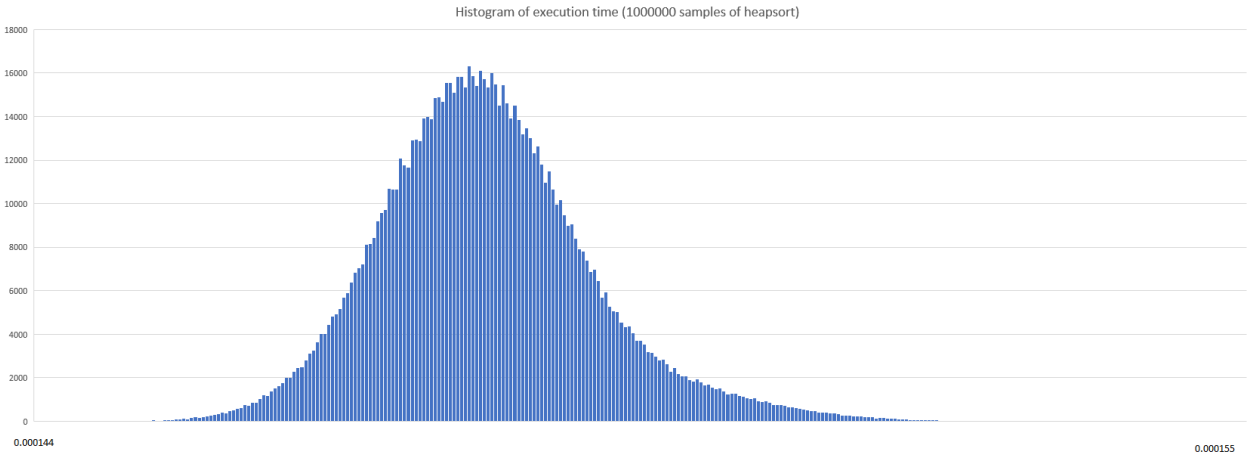

Figure 21. Histogram of execution time using Heapsort

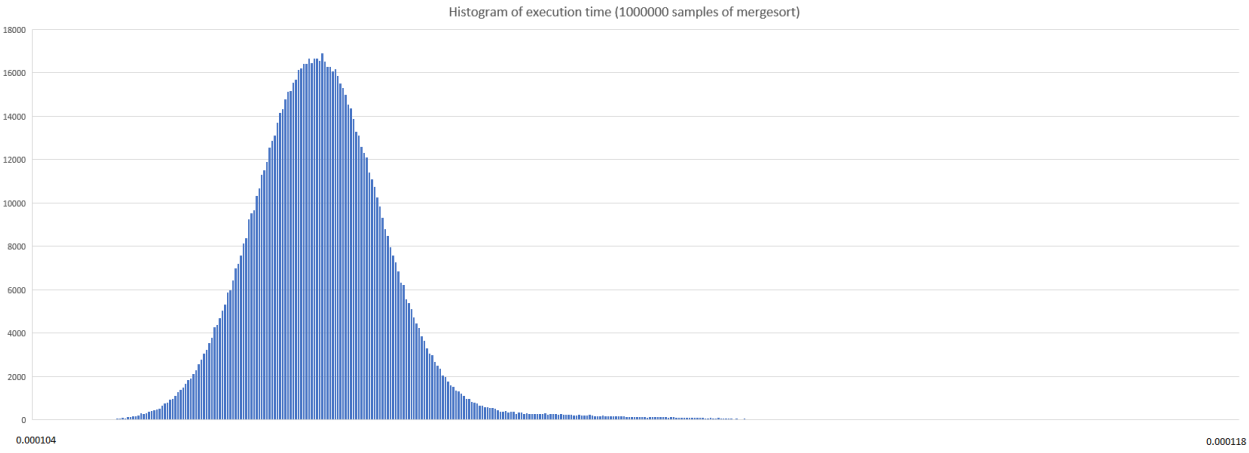

Figure 22. Histogram of execution time. Mergesort

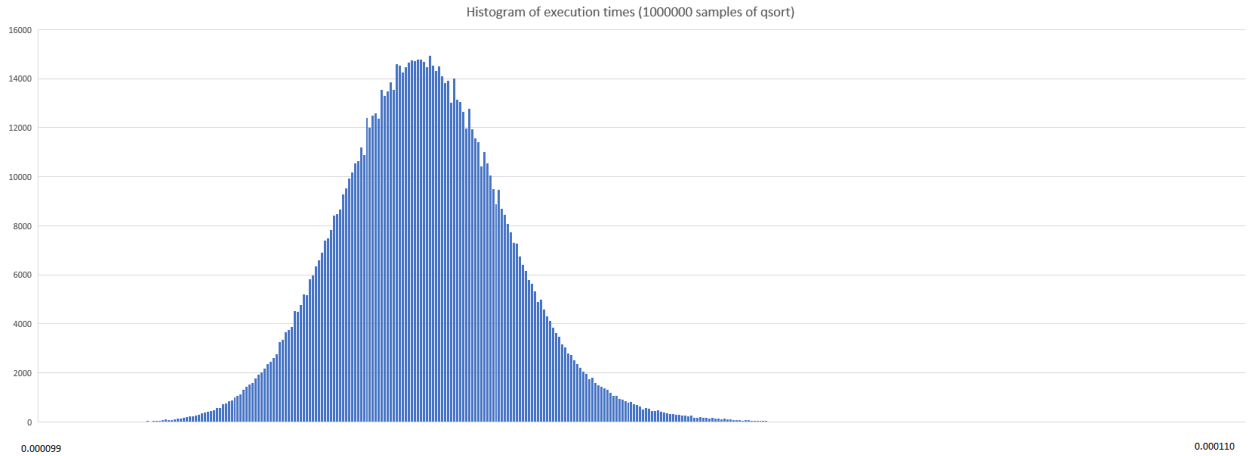

Figure 23. Histogram of execution time. Qsort

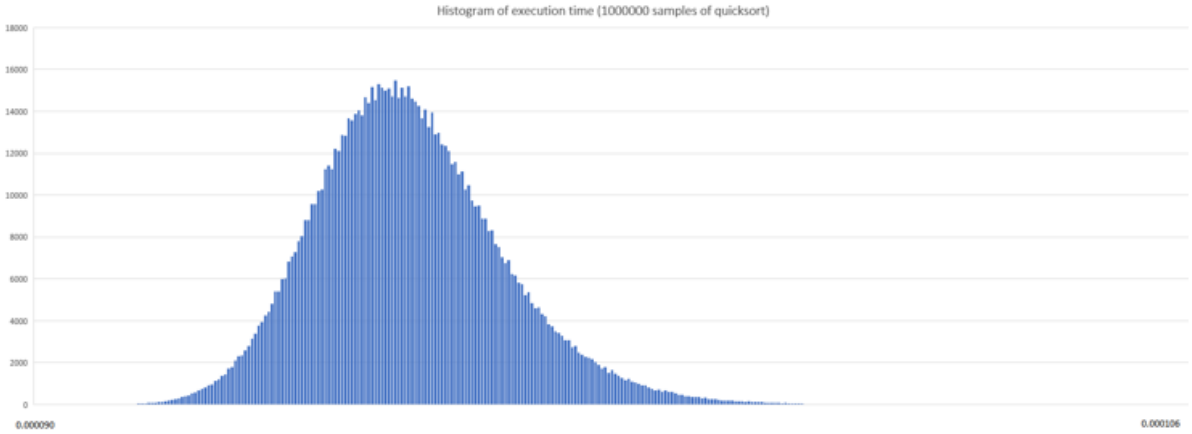

Figure 24. Histogram of execution time.

We do not find any single probability distribution that fits all target programs. For those target programs that we test, we only see unimodal distributions. As a very crude approximation, one can view these as Gaussian distributions but we find that for almost all programs that we observe, the distribution is skewed and it is not actually Gaussian. It is noteworthy that for bubblesort, the peak is skewed to the right whereas for the other target programs, the peak is skewed to the left.

4.4 Features that influence execution time

We have seen how the magnitude of execution time changes with changes in input and we have seen that for some programs, the execution time is a smooth function of input. However, with more interpretation of the program, we can better understand what influences execution time. We will now discuss this.

We will introduce a feature, called unorderedness, of input to a program, and explore how execution time of sorting algorithms depends on unorderedness. Let us start by defining unordered for a given pair. Given an input as bitvector *x*, and an integer *k*, and another integer *k'* such that *k*<*k'*, we can interpret the bitvector as an array of integers and let *k* and *k'* be indices in that vector. If the array would be sorted, then we expect that the element at index *k* would be less than or equal to the element at index *k'*. Therefore, we could say that if the element at index *k* is strictly greater than the element at index *k'*, then the pair is unordered. Thus, we define:

$$unordered(x,k,k') = \begin{cases} 1 & if\ as\_int32(x_{k*32:k*32+31}) > as\_int32(x_{k\prime*32:k\prime*32+31}) \\ 0 & otherwise \end{cases} \quad (5)$$

Then, we can compute the sum over the unorderedness over all pairs and then divide by the number of pairs. This gives us a notion of unorderedness of all pairs; another way to put it, this gives us a notion of unorderedness of the input. Thus:

$$unorderedness(x) = \frac{\sum_{k=0}^{\frac{N}{32}-1} \sum_{k'=k+1}^{\frac{N}{32}-1} unordered(x,k,k')}{\frac{\frac{N}{32} * \left(\frac{N}{32}-1\right)}{2}} \quad (6)$$

With this definition, we get that unorderedness is a number in the range [0,1] with unorderedness=0 meaning that the input is already sorted. Since a sorting algorithm rearranges the order of elements in an array, it is natural to guess that inputs with large unorderedness will cause a sorting algorithm to have long execution time. We explore this empirically for well-known sorting algorithms. The results are shown in Figure 25 through Figure 29.

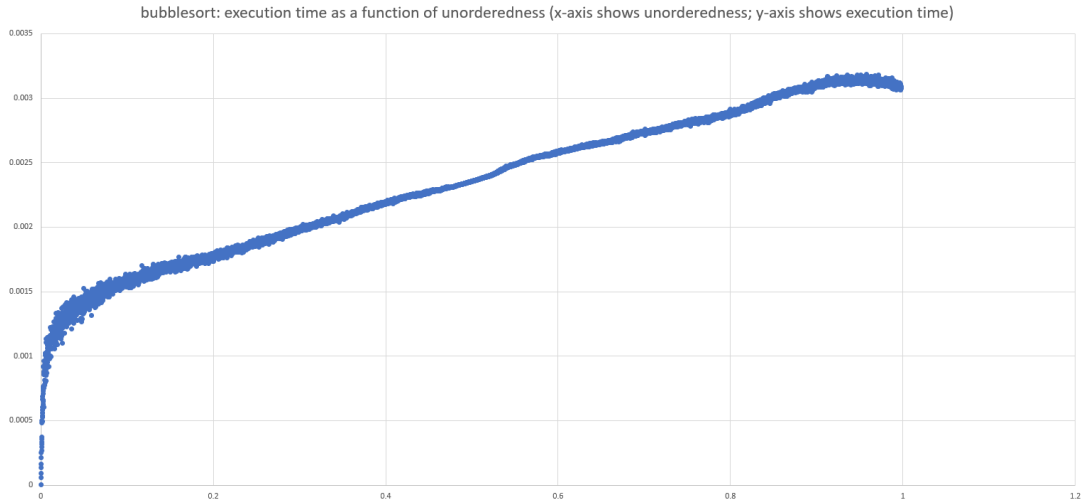

Figure 25. Bubblesort: Execution time as a function of unorderedness

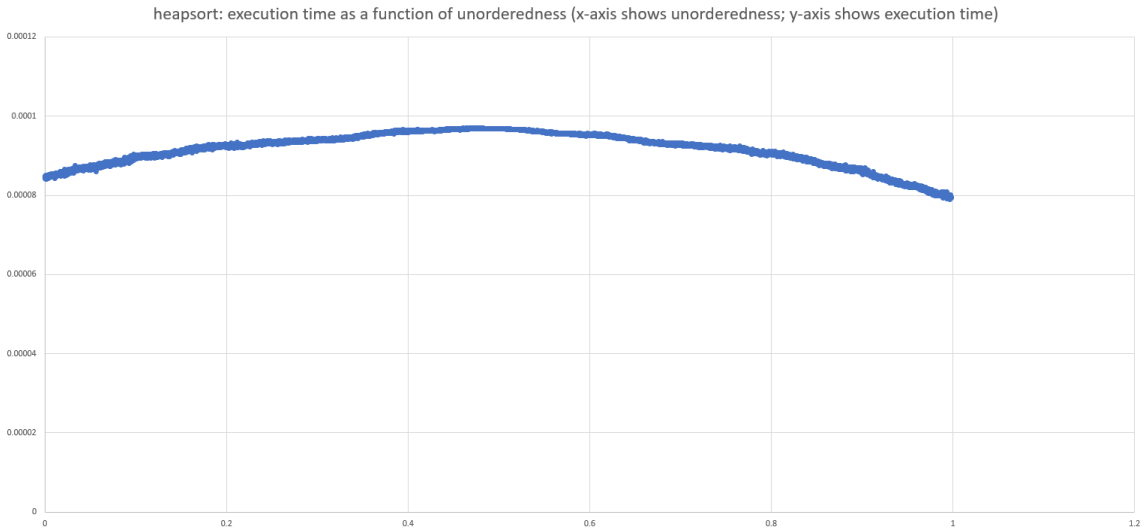

Figure 26. Heapsort: Execution time as a function of unorderedness

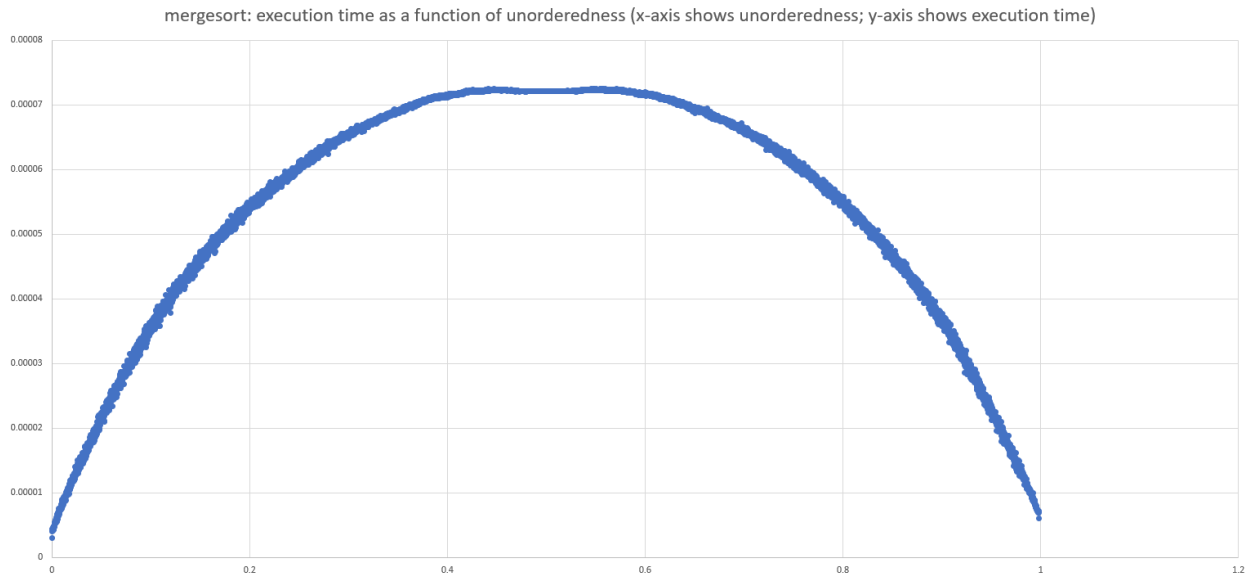

Figure 27. Mergesort: Execution time as a function of unorderedness

.

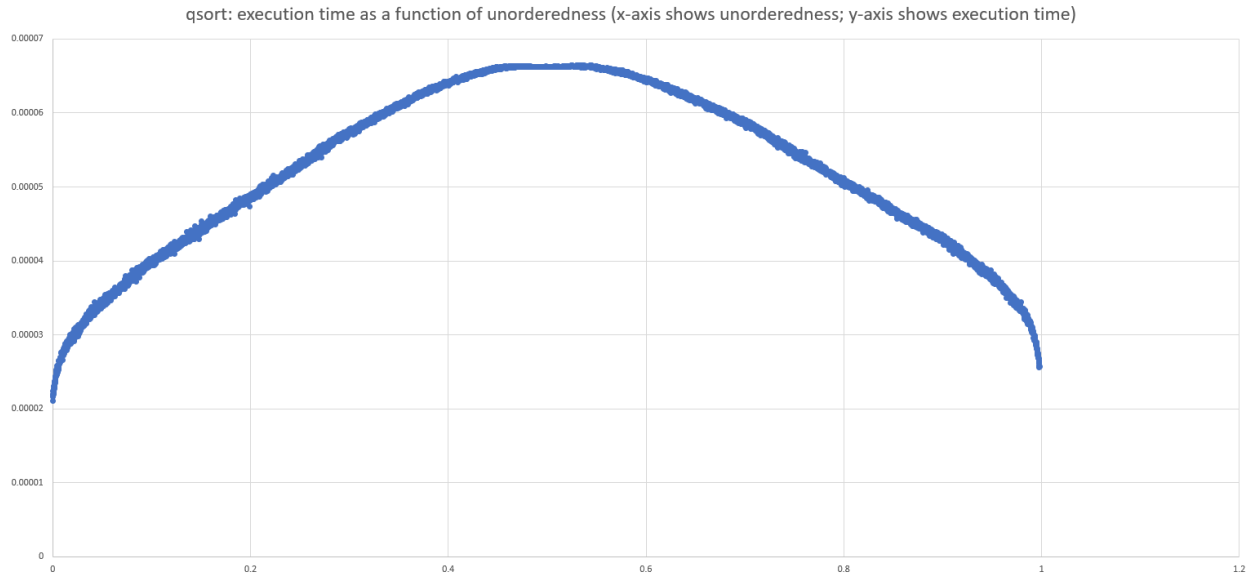

Figure 28. Qsort: Execution time as a function of unorderedness

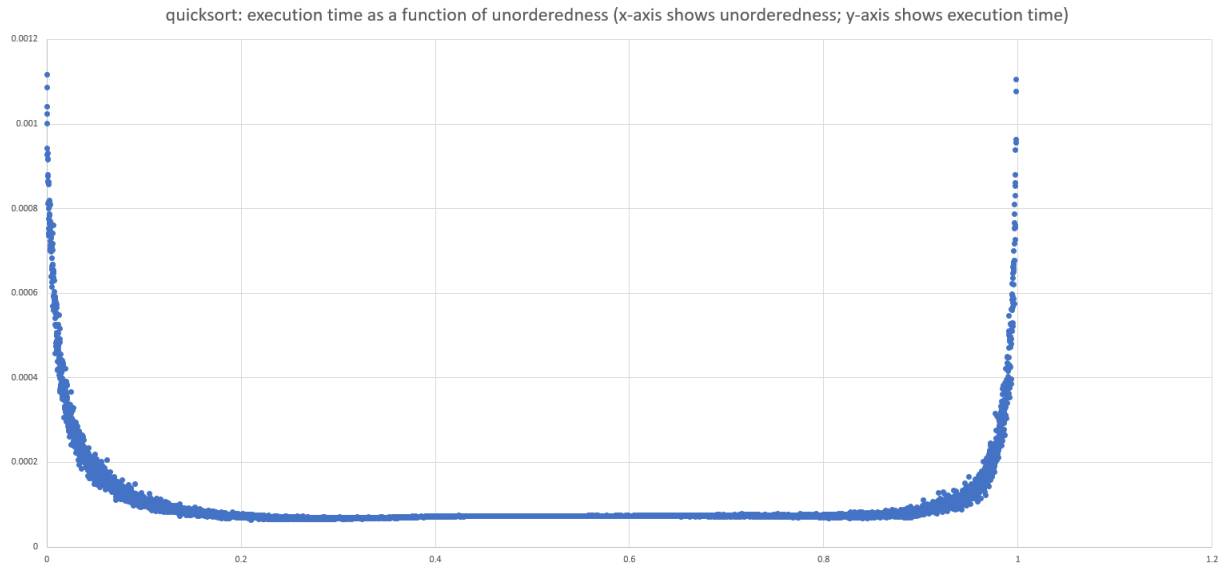

Figure 29. Quicksort: Execution time as a function of unorderedness

One can see that for Bubblesort, it holds that if unorderedness is large, then the execution time tends to be large.

Let us now explore how execution time depends on another feature. Some programs operate on floating-point numbers. The IEEE 754 standard defines both single-precision floating-point numbers (32-bit) and double-precision floating-point numbers (64-bit). Let us now discuss one issue related to that and focus on double-precision floating-point numbers. The bit layout of a double-precision floating-point number is shown in Figure 30.

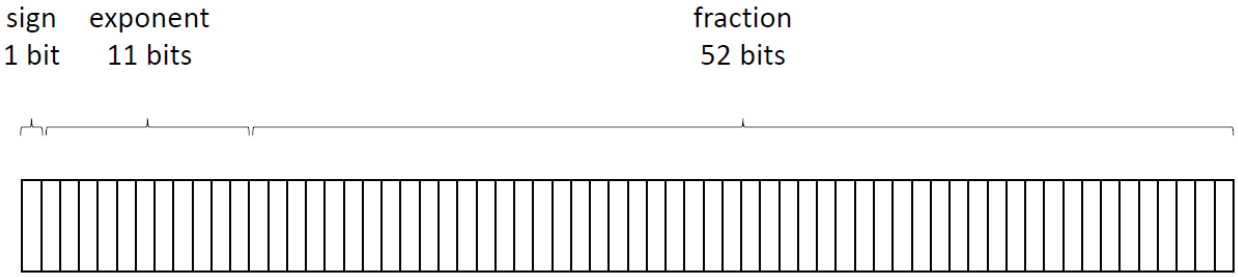

Figure 30. IEEE 754 Double precision floating point

Figure 30 shows one single bit used to indicate the sign of the number (plus or minus), 11 bits used to indicate the exponent of the number, and 52 bits used to indicate the fraction of the number. The bits used to represent the exponent are interpreted as an unsigned integer; one can then subtract an offset and this yields the exponent of the floating-point number. Hence, if the 11

bits are 00000000001, the interpreted unsigned integer of the 11 bits is 1; and the double precision floating point number has an exponent that is negative and of very large magnitude. In addition, if the 11 bits are 1111111111, the interpreted unsigned integer is very large; and the double precision floating-point number has an exponent that is positive and of very large magnitude. If the bits used to represent the exponent are all zero, then the interpreted unsigned integer is zero; this number is treated in a special way and it is called a denormal number (Figure 31).

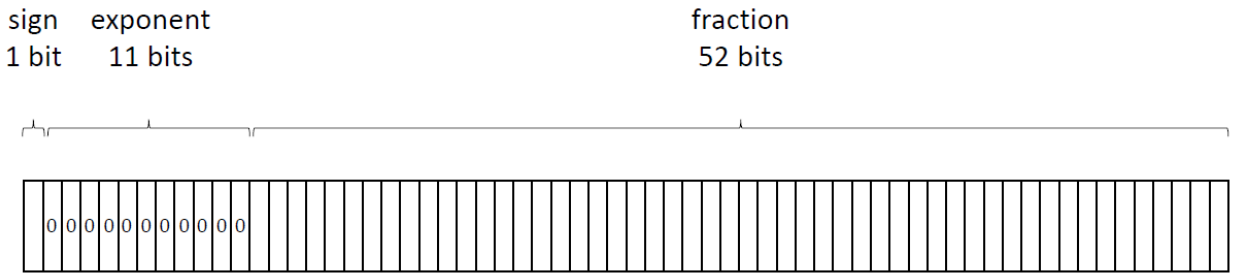

Figure 31. Denormal number in IEEE 754 Double precision floating point

Floating-point arithmetic with denormal numbers is treated differently than floating-point arithmetic with numbers that are not denormal. Typically, operations on denormal numbers are slower. Therefore, it is natural to assume that if a program takes an array of floating point numbers as input, then the execution time of the program will depend on how many of these numbers in the array are denormal. For this reason, we will now introduce definitions and measure the effect of denormal numbers.

Given a bitvector *x* with N bits such that the bitvector is interpreted as an array of double-precision floating-point numbers, let us define denormalnumber(*x,k)* as:

$$denormalnumber(x,k) = \begin{cases} 1 & if\ element\ with\ index\ k\ in\ the\ array\ represented\ by\ x\ is\ a\ denormal\ number \\ 0 & otherwise \end{cases} \quad (7)$$

Let us define fractionofdenormalnumbers(x) as:

$$fractionofdenormalnumbers(x) = \frac{\sum_{k=0}^{\frac{N}{64}-1} denormalnumber(x,k)}{\frac{N}{64}} \quad (8)$$

With this definition, we get that fractionofdenormalnumbers is a number in the range [0,1] with fractionofdenormalnumbers=0 meaning that the input has no denormal numbers. As already mentioned, it is natural to assume that if fractionofdenormalnumbers is large, then execution time is large. We explore this empirically for matrix-vector multiplication programs; each of these programs has a fixed matrix with elements in the range [0,1], takes as input a vector, and outputs a vector that is computed by multiplying the matrix by the input vector. The input vector, the output vector, and the matrix all consist of double-precision floating-point numbers. We consider two variations of this program based on how many rows the matrix has (and implicitly how many elements there are in the output matrix). The results are shown below.

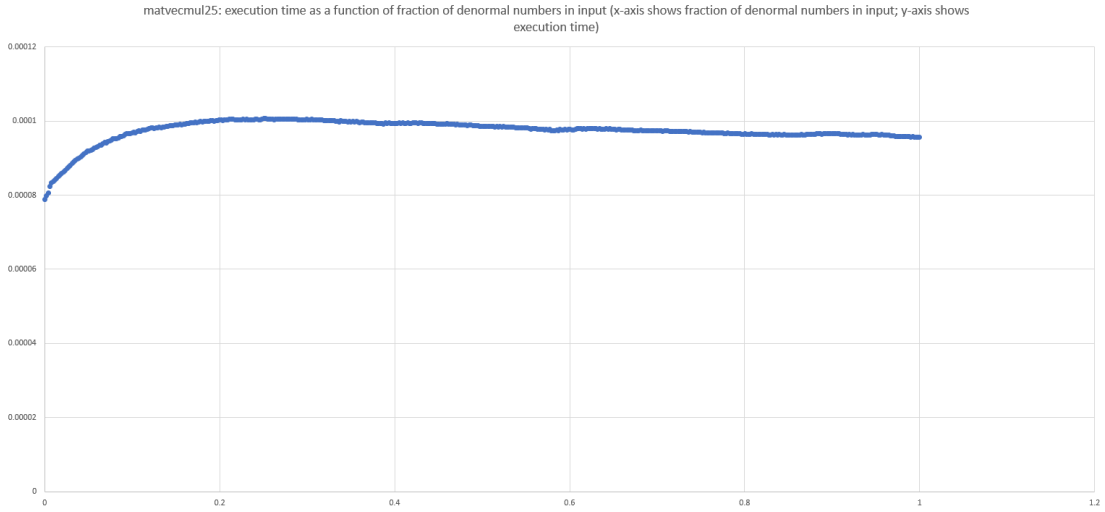

Figure 32. Matvecmul25: Execution time as a function of fraction of denormal numbers in input.

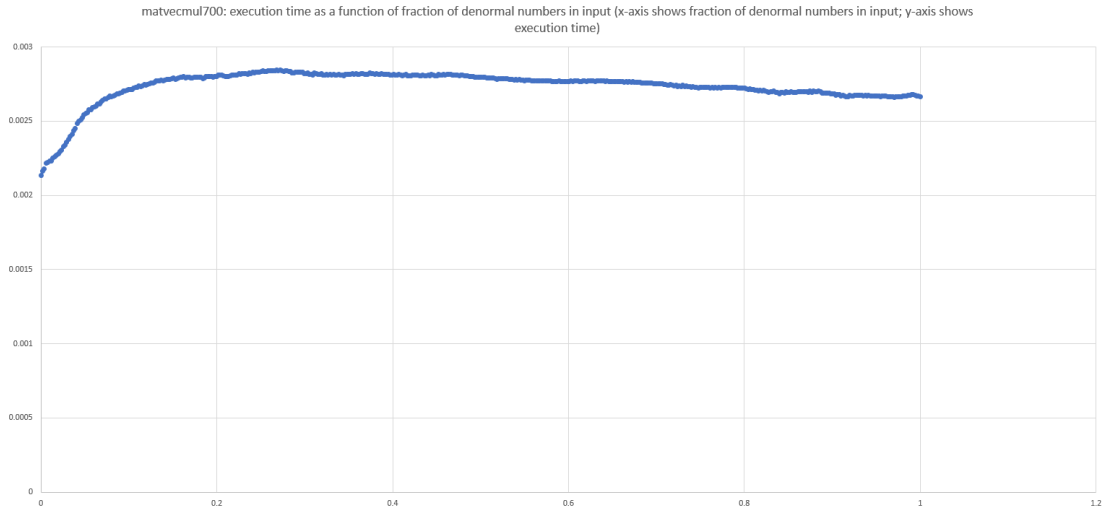

Figure 33. Matvecmul700: Execution time as a function of fraction of denormal numbers in input.

It can be seen that if the fraction of denormal numbers is larger, then the execution time is larger. However, once, there is a sufficiently large number of denormal numbers, having more denormal number do not seem to increase the execution time further. Therefore, to investigate this further, we find it interesting to explore the effect on execution time of numbers that are not denormal (i.e., numbers that are normal) but is close to denormal.

Suppose that a double-precision floating point number has the 11 bits that represent the exponents set so that the least significant bit is 1 and the other 10 bits are zero. This is shown below:

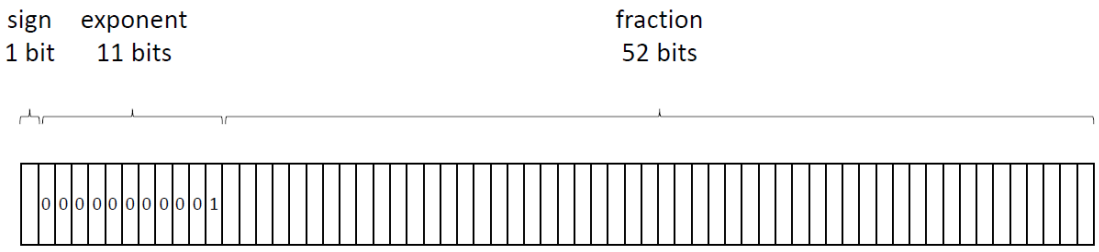

Figure 34. Normal but close to denormal in IEEE 754 double precision floating point.

Given a bitvector x with N bits such that the bitvector is interpreted as an array of double-precision floating point numbers, let us define smallestnormalnumber(x,k) as:

$$smallestnormalnumber(x,k) = \begin{cases} 1 & \text{if element with index } k \text{ in the array represented by } x \text{ is a normal number with exponent being smallest possible among normal numbers} \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

Let us define fractionofsmallestnormalnumbers(x) as:

$$fractionofsmallestnormalnumbers(x) = \frac{\sum_{k=0}^{\frac{N}{64}-1} smallestnormalnumber(x,k)}{\frac{N}{64}} \quad (10)$$

With this definition, we get that fractionofsmallestnormalnumbers is a number in the range [0,1]. It is natural to assume that if fractionofsmallestnormalnumbers is large, then because other operations are performed on these numbers, an intermediate result will be denormal. It is natural to assume that the operations whose input is a normal number but output is denormal would take extra time. It is also natural to assume that operations on normal numbers that have small exponent are more likely to yields a denormal number as a result than if the exponent were large.

Hence, it is natural to assume that if fractionofsmallestnormalnumbers is large, then execution time is large. We explore this empirically by repeating the experiment above but instead of exploring execution time as function of fractionofdenormalnumbers, we explore execution time as function of fractionofsmallestnormalnumbers. The results are shown in Figure 35 and Figure 36.

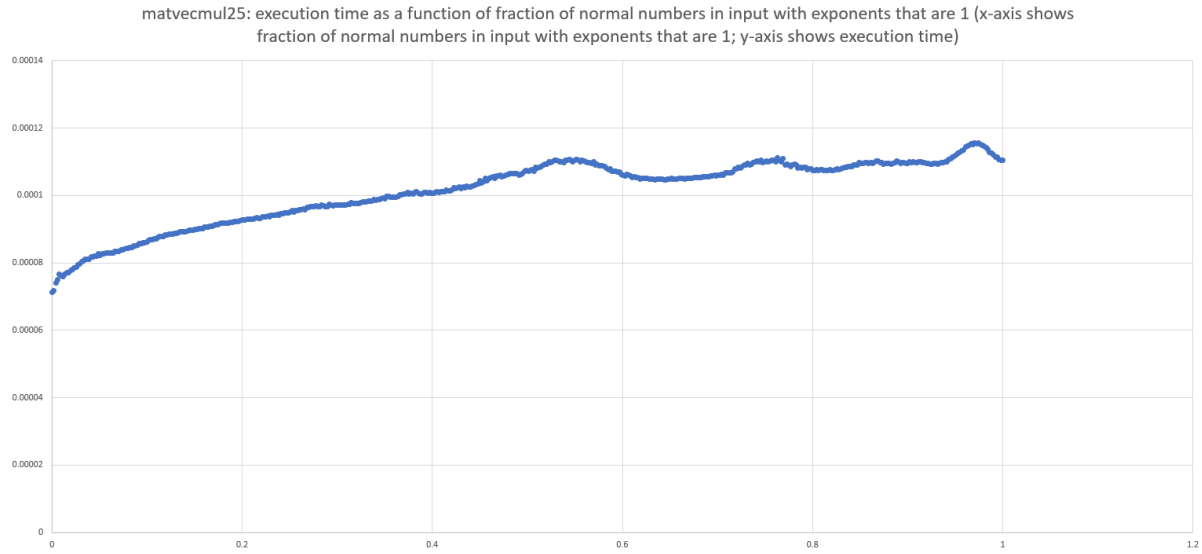

Figure 35. Matvecmul25: execution time as a function of fraction of normal numbers in input with exponents that are 1

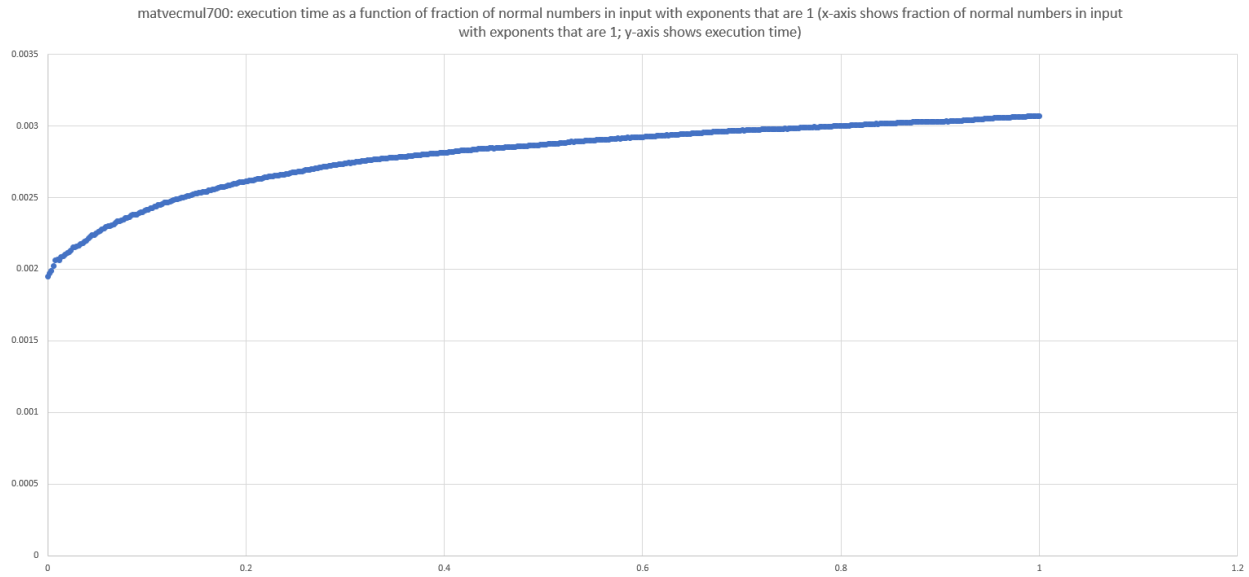

Figure 36. Matvecmul700: execution time as a function of fraction of normal numbers in input with exponents that are 1

It can be seen that if the fraction of normal numbers with exponent 1 is larger, then the execution time is larger.

We have seen that the exponent in a double-precision floating point number can influence the execution time and having seen that when this exponent is 0, the execution time is larger. For the exponent 1, the execution time is even larger. It is natural to ask whether there are other exponents that yields even larger execution times. To investigate this, we have conducted measurements whose results are shown in Figure 37 and Figure 38.

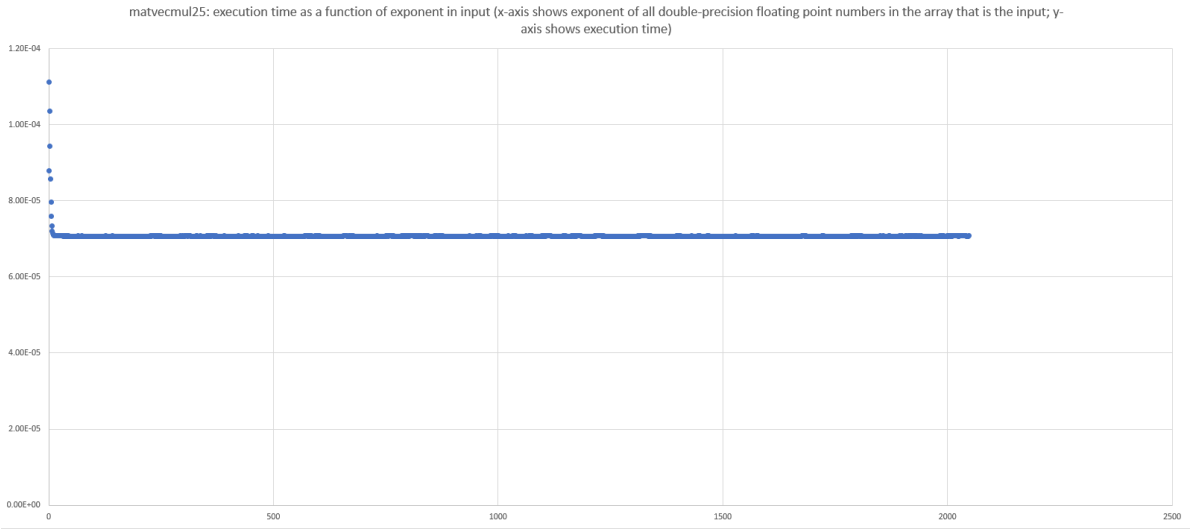

Figure 37. Matvecmul25: execution time as a function of exponent in input

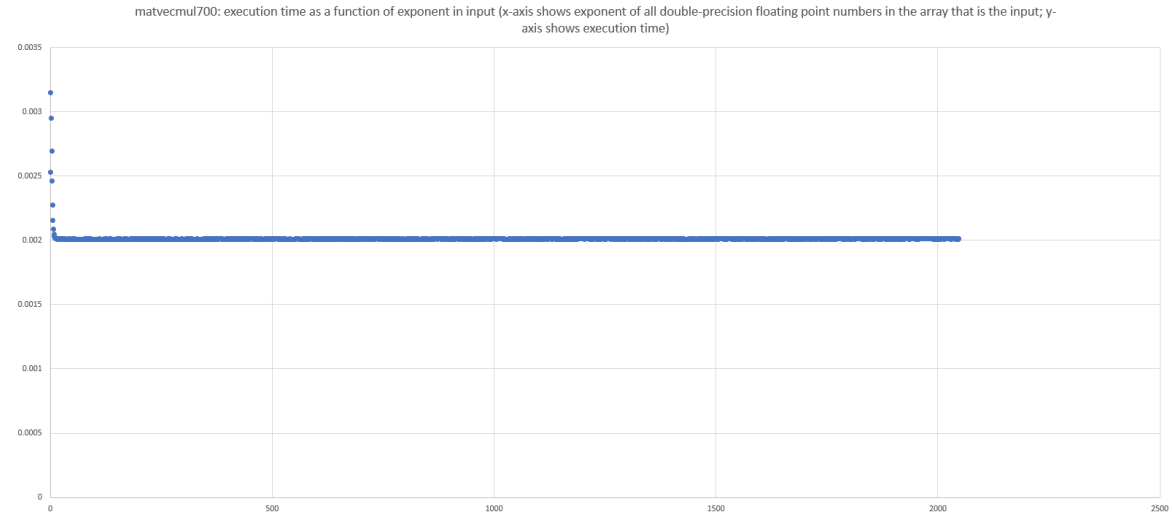

Figure 38. Matvecmul700: execution time as a function of exponent in input

It can be seen that for most exponents, the execution time is the same. However, for the exponent 0, exponent 1, and others around that, the execution time depends on the exponent. To see this more clearly, we show this as a table for relevant values below and with the largest execution time shown in red.

Table 3. Execution time depends on exponent bits in IEEE 754 double precision floating point.

For all 512 double-precision floating point numbers in the array, the bits of the exponents, interpreted as an unsigned integer is equal to:	Bit pattern of the 11 bits of the exponent	Average execution time for matvecmul25	Average execution time for matvecmul700
0	00000000000	0.000088	0.002527
1	00000000001	0.000111	0.003151
2	00000000010	0.000103	0.002947
3	00000000011	0.000094	0.002695
4	00000000100	0.000085	0.002459
5	00000000101	0.000079	0.002276
6	00000000110	0.000076	0.002158
7	00000000111	0.000073	0.002088

It can be seen that if the bits of the exponents is equal to 00000000001, then the execution time is larger.

5 Our research: New method

We have developed a new idea that uses ML for WCET analysis. Our idea is illustrated in Figure 39.

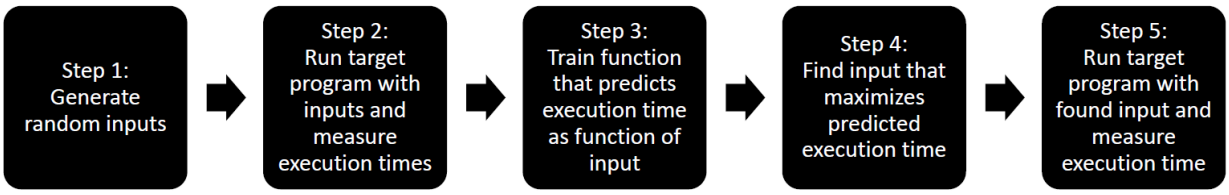

Figure 39. Overview of our idea

Our idea relies on learning and extrapolation. Specifically, the idea is as follows:

1. run the target program (i.e., the program whose WCET we seek) many times with randomly-generated inputs and measure the execution time,
2. learn a model that predicts the execution time of the target program given an input,
3. solve the optimization problem that finds the input that maximizes the predicted execution time,

4. this provides an input for which one can guess that the execution time of the target program will be large,
5. run the target program with this input and measure the execution time, and
6. this measured execution time provides a WCET estimate.

We will now elaborate on this idea. To do so, we list issues (Table 4) and then use these issues to drive the design of the method.

5.1 Issues

Table 4. Issues with new method

Issue id	1
Title:	Too few inputs with extreme execution times
Description:	When running target program with random inputs, the histograms of many of these programs tend to be Gaussian or Gaussian-like. Thus, there are many runs where execution times are non-extreme and very few runs where the execution times are extreme (very small or very large). Hence, it can happen that the training will be dominated by runs that have non-extreme execution times. This can cause the magnitude of the training error to be small for those training examples that are non-extreme and large for the training examples that are extreme. Thus, for training examples with a large execution time, the learned function does not offer good prediction. This can be problematic because we want to detect patterns in input that causes long execution times and accentuate these patterns. This does not work if the training is not influenced by runs with large execution time.
Step where this is an issue	Step 1
Issue id	2
Title:	Rare non-repeatable increase in execution time
Description:	As seen earlier in this report, there are rare non-repeatable large (0.8ms) deviations in execution times. These can occur when a target program is run many times with the same input.

Step where this is an issue	Step 2
Issue id	3
Title:	Function learned is not sufficiently expressive
Description:	When training a function on training examples, one typically selects a function with weights and assigns values to these weights so that one obtains a function that fits the training examples. If the function is not sufficiently expressive (too few weights or the structure of the description of the function is not sufficiently expressive), then the obtained function may not fit the training examples.
Step where this is an issue:	Step 3
Issue id	4
Title:	Training takes too long
Description:	Training is typically formulated as an optimization problem where one formulates a loss function that measures how good fit is achieved against the training examples. We have tried standard packages for curve fitting and mathematical optimization to solve this exactly, but this takes too long (did not terminate).
Step where this is an issue:	Step 3
Issue id	5
Title:	Training requires too much main memory
Description:	In the experiments that we conducted for our method, for each target program, there are 30 million inputs and each input is 32768 bits. This consumes 132 GB of memory. If each bit would be represented by a single-precision floating point number (which is common in many machine learning packages), then it would consume 32 times as much memory; that is, 4TB of memory. Most computers today do not have this amount of main memory. Hence, it is important to perform training so that not too much main memory is required. It has been our experience that storing training examples on disk yields a very

	large increase in the time required for training (hence making Issue 4 more severe).
Step where this is an issue:	Step 3
Issue id	6
Title:	Training yields a function that does not generalize.
Description:	Overfitting is a well-known issue in machine learning.
Step where this is an issue:	Step 3
Issue id	7
Title:	Training yields a function that cannot extrapolate
Description:	Even if the training fits all training examples and can generalize, there is still an issue with the potential inability to extrapolate. Suppose that we train a function so that its training error is low and its validation error (the error on examples that were not part of the training set) is low on average. Suppose that the learned function is such that for all inputs, the predicted execution time is never greater than the execution time of the training example with the largest execution time. In this case, Step 4 of our method cannot find an input with larger predicted execution time.
Step where this is an issue:	Step 3
Issue id	8
Title:	Finding the input that maximizes the predicted execution time takes too long.
Description:	Finding the input that maximizes the predicted execution time involved solving an optimization problem. Some optimization problem takes a long time to solve—often the solver does not terminate.
Step where this is an issue:	Step 4

Issue id	9
Title:	The memory mapping of Step 2 is different from the memory mapping of Step 5.
Description:	The execution time of a target program depends on the memory mapping (mapping from virtual-addresses to physical addresses). If the same program is run many times, and if this is started as a new process for each run, then the memory mappings may be different for each run. The reason why the execution time of a target program depends on memory mapping is that for each run, the target program generates virtual addresses and they are the same for each run. However, these virtual addresses are translated to physical addresses and these physical addresses determine how resources in the memory system are accessed (e.g., which cache set is accessed). Because of this, one potential issue is that if the memory mapping in Step 2 is different from the memory mapping in Step 5, then the learned function in Step 3 may only apply to the memory mapping in Step 2 and hence this will not help us in finding an input that causes long execution time in Step 4. Hence, when we run the target program with the input that we believe will have long execution time (Step 5), it may not have long execution time.
Step where this is an issue:	Step 2 and Step 5
Issue id	10
Title:	The WCET estimation produced applies only to one memory mapping.
Description:	If the target program when deployed in an aircraft is run with different memory mappings after each boot (e.g., because it is started by a script), then it may happen that it runs with different memory mappings and these will have different execution times.
Step where this is an issue:	Step 2 and Step 5

5.2 Resolutions of the issues identified in Table 4

We do not address issue 10 in this report because it is beyond WCET analysis; it is related to the run-time system and linking. We address the other issues below (Table 5)..

Table 5. Issue resolution

Issue id	1
How we deal with it:	There are three ways we deal with it: • Pruning: Given n training examples, we select M of them (where M<n). This is achieved as follows. Among the n training examples, select (i) the 0.45*M ones with largest execution time, (ii) the 0.45*M ones with the smallest execution time, and (iii) 0.1*M training examples from the other. In this way, we have selected M training examples from the n training examples and these M training examples have more extreme execution times. Hence, we use more extreme training examples for training. • Weighting: Assign a weight to each training example. If the training example i is assigned weight Q, then when performing the training, we treat it as if there were Q identical training examples of training example i. Hence, if Q is large, then this training example has more influence on the training. In this way, we can assign a higher weight to training examples with extreme execution times. • Min max magnitude of error: Perform training with the loss function that is maximum magnitude of error among training examples. This is different from what is common in machine learning: minimize sum of square of error.
Issue id	2
How we deal with it:	Let NREPL denote an integer. If we want to generate n random inputs, then for each of these inputs, run the target program NREPL times and measure the execution time for each of these runs. Then, for each input, compute the median execution time among the runs with this input. Then, we treat this median as the single execution time for

	this input. These are then fed into the training procedure in Step 3. Note that this makes the training insensitive to disturbances.
Issue id	3
How we deal with it:	We start by training an affine function. Assume that N denotes the number of bits of the input to the target program, x_j denotes the j:th bit in the input, and f denotes the function that predicts the execution time. Then this affine function is expressed as: $$f(x_0, x_1, \dots, x_{N-1}) = \sum_{j=0}^{N-1} w_j * x_j + w_N$$ We train this function; this involves assigning values to the w_j parameters. This function is not very expressive. However, after we have done this training, we compute the error for each training example. Then, we select the training examples with largest magnitude of error. Then, we iterate over all pairs of bits in the input (that is N*(N-1)/2 pairs) and for each combination of values (0 or 1) that can be assigned to these two bits; for each pair, for each combination of values, we check if this occurs frequently among the training examples with large magnitude of error. If so, then we decide that this combination should form a function which we call basis function (we call it basis function even if it does not follow the mathematical definition of a basis function) and this gets added to the function that we want to train. Thus, this gives us the following $$f(x_0, x_1, \dots, x_{N-1}) = \sum_{j=0}^{N-1} w_j * x_j + w_N + \sum_{k=0}^{NBASISFUNCTIONS-1} v_k * \varphi_k\left(x_{bitindex1_k}, x_{bitindex2_k}\right)$$ where $$\varphi_k\left(x_{bitindex1_k}, x_{bitindex2_k}\right) = \begin{cases} 1 & if\ x_{bitindex1_k} = bitvalue1_k\ and\ x_{bitindex2_k} = bitvalue2_k \\ 0 & otherwise \end{cases}$$

	where basis function k is given the name φ_k. It is defined by five parameters $bitindex1_k$, $bitindex2_k$, $bitvalue1_k$, $bitvalue2_k$ and v_k. We can now train f by selecting values of w_j and v_k. A simpler version of this is to not select the basis functions from the training examples with largest magnitude of error but instead generate the parameters $bitindex1_k$, $bitindex2_k$, $bitvalue1_k$, $bitvalue2_k$ randomly (v_k is still trained though). This has the benefit that training is faster but the training gets worse fit. As can be seen later in the evaluation section of this report, although we attempted to deal with this issue, this was not enough. We actually need even more expressive functions because for many target programs the execution time as a function of input is not smooth.
Issue id	4
How we deal with it:	Training is typically achieved by minimizing a loss function. We deal with the issue that training takes too long by solving this optimization problem approximatively. It works as follows. The initialization is to assign values to weights in some way (randomly or in some conscious way). Compute the error for each training example, and then compute the gradient of the loss function with respect to the weights. This yields a direction in which the loss function increases the most. Since we want to decrease the loss function, we change the weights in the opposite direction of this computed gradient. Determining the direction of change to the weights is not enough; we also need to determine the magnitude of change to the weights. This is determined by a parameter called α. Typically, in machine learning, α =0.01. We have found that in our application, we need much smaller value of α though. We have also found that when we try to minimize the loss function where the loss function is maximum magnitude of error (as opposed to sum of square of error), then it is crucial to make two changes: First, select α that is smaller and be adaptive. Therefore, when we minimize the loss function where the loss function is maximum magnitude of error, we try different values of α and only

	make a change if the change results in a lower loss function. Second, the gradient should not be computed based on all training examples but only based on the k training examples with the largest magnitude of error, where we vary k in {1..15}.
Issue id	5
How we deal with it:	We store all training examples in main memory. In order for this to be possible, we do not represent each bit as a floating point number; a bit is stored as a bit. But when we compute the gradient of a training example, we convert its bits to floating point numbers and compute the gradient. However, this is only needed for one training example at a time so its extra memory required is very small. Since we store all training examples in main memory, we can also quickly create an index that provides the training examples in descending sorted order of magnitude of error; this is helpful for dealing with issue 3.
Issue id	6
How we deal with it:	To avoid overfitting, we make sure that the number of weights in the function f is much smaller than the number of training examples. We use 30 million training examples but the number of weights is less than one million.
Issue id	7
How we deal with it	To avoid the issue that training yields a function that cannot extrapolate, the following is noteworthy. First, the affine function can extrapolate. Second, when we use basis functions on two bits, we are not sure whether extrapolation is always possible; this may deserve further study.
Issue id	8
How we deal with it:	To deal with the issue that finding the input that maximizes the predicted execution time takes too long, we use two techniques. First, if the function is affine, then solving this optimization problem can be done quickly as follows. If $w_j>0$, then set x_j to 1; otherwise set x_j to 0.

	Second, if the function is not affine but it is a function that is affine plus the sum of basis functions based on two bits as mentioned earlier, then we can get an input by using the first approach and then improve it iteratively. The iteration is as follows. For each basis function, consider its two bits and consider all four combinations of values to these two bits and for each combination, evaluate the function; then assign values to these two bits based on which yields the largest value of the function. This can be iterated many times and it leads to larger value of f (though not necessarily the maximum).
Issue id	9
How we deal with it:	To deal with the issue that the memory mapping of Step 2 is different from the memory mapping of Step 5, we simply make sure that there is no difference. Specifically, it is done as follows. Our WCET analysis tool is linked together with the target program. When our WCET analysis tool performs Step 5, the memory mapping is the same as when it performs Step 2.
Issue id	10
How we deal with it:	We do not deal with this issue. One could re-run our WCET analysis tool many times and then this will yield different WCET estimates of the target program for different memory mappings. One can then take the maximum among them and treat it as a WET estimate.

6 Our research: Tool

We have implemented a tool based on this idea. The tool is written in C code and runs on Linux. It uses parallel processing and performs the learning with all data in memory. The latter is possible because we use a computer with 256GB of memory to store the 30 million training examples.

During our research, we have developed many tools. Here, we present two of them. We call them Version 1 and Version 2 where Version 2 is an improvement of Version 1.

They deal with all the issues mentioned earlier (except Issue 10) but they deal with Issue 1 and 3 in different ways.

6.1 Version 1

Version 1 of the tool uses the pruning technique to deal with Issue 1. For this pruning, it uses M=2000000. To deal with Issue 3, it uses basis functions that are randomly generated; for this, it uses NBASISFUNCTIONS=1000000.

6.2 Version 2

Version 2 differs from version 1 in two important ways. First, the loss function used during training is the maximum magnitude of error. Second, to deal with Issue 3, it uses basis functions but these basis functions are not randomly generated; instead, these basis functions are selected to be the ones that help to decrease the magnitude of training error. These basis functions are generated as follows:

- After completing the training of affine function, we iterate over the training examples to identify the training examples that have large magnitude of error; here error is computed as the difference between the actual value minus the predicted value from the affine function. The number of training examples is 30 million but the number of training examples that we select is NEXAMPLES_FOR_FINDING_BASISFUNCTIONS=30000.
- Then, we iterate over all 4-tuples <bitindex1, bitindex2, bitvalue1, bitvalue2> where bitindex1 is bitindex of the input to the program; bitindex2 is bit index of the input to the program; bitvalue1 in {0,1}; bitvalue2 in {0.1} such that bitindex1 < bitindex2. Each 4-tuple represents a basis function meaning that if the input to the target program is such that the bit with index bitindex1 is bitvalue1 and the bit with index bitindex2 is bitvalue2, then the basis function outputs 1; otherwise the basis function outputs 0.

In our experimental evaluation, we have target programs with 32768 bits as input and hence we obtain approximately 2 billion basis functions. The number of these basis functions is too large. Therefore, we want to select a subset of these basis functions. We evaluate each of these approximately 2 billion basis functions on each of the NEXAMPLES_FOR_FINDING_BASISFUNCTIONS = 30000 training examples. Then, for each of these basis functions, we count how many times it evaluated to 1. Then, we sort the basis functions in descending order of their count of evaluation to 1. In this way, we have 2 billion basis functions and they are sorted in descending order of how often they evaluate to 1 for training examples where we currently have large magnitude of error. That is, we have 2 billion basis functions and they are sorted in order of how much they can help to decrease the magnitude of training error. We select the NBASISFUNCTIONS=10000 first basis functions from this

order. This gives us NBASISFUNCTIONS=10000 that help to decrease the magnitude of training error.

7 Our research: Evaluation

We have tested out tool on a suite of benchmark programs that we have developed. We find that our approach performs well on bubblesort, matvecmul25, and matvecmul700. The latter two perform multiplication of a matrix by a vector where the matrix is fixed and the vector is the input. Details are provided below for Version 1 and Version 2 respectively.

7.1 Version 1

This section presents an evaluation of our method on a set of target programs for one specific setting; it is the setting NBASISFUNCTIONS=1000000 and M=2000000. The data is shown in Table 6.

It can be seen that our approach is effective for bubblesort, matvecmul25, and matvecmul700. The reason why our approach is effective for bubblesort is that for bubblesort, the execution time becomes long if there is a need to do many sweeps and this occurs when the numbers that are sorted by bubblesort are in approximately descending order in the array. Our approach can also pick up bit patterns in the input that indicate approximately descending order of the array. The number of sweeps performed by bubblesort is not the only factor that influences the execution time though, the performance of branch predictor matters as well. If the numbers that are sorted are in descending order, then the number of sweeps is large but the performance of the branch predicted is very good and hence the execution time is not maximized. The execution time is maximized when the number of sweeps is large and there is enough “chaos” in the data so that the branch predictor performs poorly. This occurs for inputs where the numbers are in approximatively descending order. Our method notices both of these effects and decides that the input for which one can guess the execution time is large is an input that is approximatively in descending order.

The reason why our approach is effective for matvecmul25 and matvecmul700 is that arithmetic operations on denormal numbers (https://en.wikipedia.org/wiki/Denormal_number) are slower than other numbers and our approach detects that and constructs input to the program so that the program has to perform more of those operations.

Table 6. Performance of WCET analysis. Version 1. Output.

		Performance of WCET analysis			
	[mint,maxt]	WCET estimate		WCET estimate / maxt	
		Only affine	With basis functions	Only affine	With basis functions
BUBBLESORT	[0.001985610, 0.002679730]	0.002707772	0.002700042	1.010464487	1.007579868
QUICKSORT	[0.000065130, 0.000081410]	0.000058219	0.000062760	0.715133276	0.770912664
QSORT	[0.000064910, 0.000073370]	0.000057520	0.000058949	0.783971651	0.803448276
HEAPSORT	[0.000088070, 0.000104480]	0.000089639	0.000090349	0.857953675	0.864749234
MERGESORT	[0.000068540, 0.000077930]	0.000063249	0.000065189	0.811612986	0.836507122
MATVECMUL25	[0.000068730, 0.000079910]	0.000086819	0.000070639	1.086459767	0.88398198
MATVECMUL700	[0.001922460, 0.002238300]	0.002380286	0.001951602	1.06343475	0.871912612
FFT_RECT	[0.000115860, 0.000136100]	0.000116229	0.000116228	0.853997061	0.853989713
FFT_POL	[0.000129760, 0.000156570]	0.000143058	0.000142638	0.913699943	0.911017436
FFT_REALONLY	[0.000271150, 0.000313680]	0.000271976	0.000271896	0.867049222	0.866794185

In order to understand how good fit we get from the learning for these methods, we present the data in Table 7.

Table 7. Performance of WCET analysis. Version 1. Training.

		Training					
	[mint,maxt]	Average abs error		Max abs error		Prediction range	
		Only affine	With basis functions	Only affine	With basis functions	Only affine	With basis functions
BUBBLESORT	[0.001985610, 0.002679730]	0.000105562	0.000099193	0.000514151	0.000470991	[0.001810560, 0.002876299]	[0.001871325, 0.002874758]
QUICKSORT	[0.000065130, 0.000081410]	0.000002331	0.000001814	0.000011395	0.000009618	[0.000068032, 0.000072668]	[0.000064944,0.000075783]
QSORT	[0.000064910, 0.000073370]	0.000001187	0.000000927	0.000005130	0.000004233	[0.000066282, 0.000069801]	[0.000065125,0.000071103]

		Training					
	[mint,maxt]	Average abs error		Max abs error		Prediction range	
		Only affine	With basis functions	Only affine	With basis functions	Only affine	With basis functions
HEAPSORT	[0.000088070, 0.000104480]	0.000003066440724150	0.000002546609672704	0.000009088	0.000009332	[0.000092475, 0.000097778]	[0.000089089, 0.000100348]
MERGESORT	[0.000068540, 0.000077930]	0.000001714	0.000001329	0.000005280	0.000004968	[0.000071608, 0.000074379]	[0.000069251,0.000077164]
MATVECMUL25	[0.000068730, 0.000079910]	0.000000808	0.000000629	0.000009825	0.000007655	[0.000069058, 0.000070445]	[0.000067872,0.000072534]
MATVECMUL700	[0.001922460, 0.002238300]	0.000091637	0.000070610	0.000206496	0.000238863	[0.001960951, 0.002087486]	[0.001840231,0.002226342]
FFT_RECT	[0.000115860, 0.000136100]	0.000004536	0.000003468	0.000015146	0.000014686	[0.000117812, 0.000124055]	[0.000111637,0.000130349]
FFT_POL	[0.000129760, 0.000156570]	0.000002948	0.000002469	0.000020023	0.000016461	[0.000122991, 0.000156733]	[0.000123418,0.000155368]
FFT_REALONLY	[0.000271150, 0.000313680]	0.000008251	0.000006404	0.000026219	0.000026241	[0.000281148, 0.000291890]	[0.000267186,0.000303852]

In order to understand how good this learning is for extrapolation, we present the data in Table 8.

Table 8. Performance of WCET analysis. Version 1. Extrapolation.

		Find input so that predicted execution time is large			
	[mint,maxt]	Predicted execution time for this input		Measured execution time for this input	
		Input is obtained by maximizing affine function	Input is obtained by maximizing function with basis functions	Input is obtained by maximizing affine function	Input is obtained by maximizing function with basis functions
BUBBLESORT	[0.001985610, 0.002679730]	0.008551431	0.013720491	0.002707772	0.002700042
QUICKSORT	[0.000065130, 0.000081410]	0.000119306	0.000267713	0.000058219	0.000062760
QSORT	[0.000064910, 0.000073370]	0.000103930	0.000176736	0.000057520	0.000058949
HEAPSORT	[0.000088070, 0.000104480]	0.000165705	0.000302052	0.000089639	0.000090349
MERGESORT	[0.000068540, 0.000077930]	0.000110767	0.000216517	0.000063249	0.000065189
MATVECMUL25	[0.000068730, 0.000079910]	0.000089443	0.000141487	0.000086819	0.000070639

MATVECMUL700	[0.001922460, 0.002238300]	0.003918258	0.009542136	0.002380286	0.001951602
FFT_RECT	[0.000115860, 0.000136100]	0.000208289	0.000485065	0.000116229	0.000116228
FFT_POL	[0.000129760, 0.000156570]	0.000263253	0.000483781	0.000143058	0.000142638
FFT_REALONLY	[0.000271150, 0.000313680]	0.000449975	0.000968339	0.000271976	0.000271896

In order to understand how time is spent in the WCET analysis, we present the data in Table 9. We report it only for bubblesort. The same number applies to the other target programs as well (except the time spent to collect execution time, which is shorter for other programs because they have smaller execution times)

Table 9. Time required for WCET analysis. Version 1.

	Time spent to collect execution times for 30 million runs	**Time spent to obtain 2 million training examples from the 30 million runs**	**Time spent to train affine function**	**Time spent to train basis functions**	**Time spent to find input that yields large predicted execution time (for the case with basis functions)**
BUBBLE-SORT	REPL*30000000 0.0023 seconds = 4 days	5 minutes	21 minutes	14.5 hours	1.5 hours

It can be seen that our approach, which learns the execution time as a function of inputs using an affine function, offers a performance advantage as compared to the brute force approach (using maxt) for some target program (bubblesort, matvecmul25, matvecmul700). However using the basis functions do not offer much of an advantage for other target programs.

7.2 Version 2

We evaluated Version 2 on the target program bubblesort using the aforementioned experimental setup. Table 10 through Table 12 list the results.

Table 10. Performance of WCET analysis. Version 2. Output.

		Performance of WCET analysis			
	[mint,maxt]	**WCET estimate**		**WCET estimate / maxt**	
		Only affine	With basis functions	Only affine	With basis functions
BUBBLE-SORT	[0.001985610, 0.002679730]	0.002707139	0.002706180	1.010228269	1.009870397

Table 11. Performance of WCET analysis. Version 2. Training.

		Training					
	[mint,maxt]	Average abs error		Max abs error		Prediction range	
		Only affine	With basis functions	Only affine	With basis functions	Only affine	With basis functions
BUBBLESORT	[0.001985610, 0.002679730]	0.000076717	0.000076695	0.000251554	0.000247332	[0.002164713, 0.002469170]	[0.002164563, 0.002468843]

Table 12. Performance of WCET analysis. Version 2. Extrapolation.

		Find input so that predicted execution time is large			
	[mint,maxt]	Predicted execution time for this input		Measured execution time for this input	
		Input is obtained by maximizing affine function	Input is obtained by maximizing function with basis functions	Input is obtained by maximizing affine function	Input is obtained by maximizing function with basis functions
BUBBLESORT	[0.001985610, 0.002679730]	0.004345720	0.004345885	0.002707139	0.002706180

7.3 Comments

We see that our method performs well on bubblesort, matvecmul25, and matvec700. For other target programs, it does not perform well. The reason is (i) for these programs, the execution time as a function of input is not a smooth function, and (ii) our function is not expressive enough.

8 Conclusions

Many parts in aircraft today rely on software that interacts with its physical environment. Typically, this interaction involves taking sensor readings, sending actuation commands, reading commands from humans, and presenting information to humans. These interactions require that software deliver results at the right time, as argued in the guidance document DO-178C and in previous FAA reports. Correct timing, in turn, depends on the execution time of individual programs. Hence, the problem of finding an upper bound on the execution time of a

program, called Worst-Case Execution Time (WCET) analysis, is an important step in avionics certification.

Unfortunately, WCET analysis is difficult for several reasons. Typically, a program can execute a large number of paths. During the execution of one path, the program uses resources in a complex way and this resource use is difficult to analyze. Finally, during the execution of one path, the speed of execution depends on temperature, which, in turn, depends on earlier execution.

As a result, the state-of-the-art and state-of-the-practice in WCET analysis today have limitations in terms of (i) the maximum size of the program, (ii) the complexity of the hardware, and (iii) the ability to analyze hardware that does not have documentation available.

Thus, it is worth exploring ideas to overcome these limitations. The disciplines of machine learning (ML) and artificial intelligence (AI) provide general methods for obtaining knowledge from observations; and for using knowledge for prediction and decision-making; and doing so in uncertain, unknown, or complex environments. These general methods appear to hold promise for dealing with the limitations of current WCET analysis methods.

This report has assessed the use of machine learning (ML) and artificial intelligence (AI) to find the WCET of avionics software. This report includes (i) background to the problem and challenges, and (ii) a new method (based on ML) that we have developed for WCET analysis and a tool based on this method; we call the method “learn-and-extrapolate.”

Our findings are as follows:

1. Our new WCET analysis method performs well for some target programs in the sense that it can find an input that causes long execution time caused by counter-intuitive effects that normal WCET analysis tools do not consider.
2. There are target programs for which our new WCET analysis method does not perform well.
3. Hence, our WCET analysis method is a complement to other WCET analysis methods.
4. For our WCET analysis method, how to measure execution time is critical. We have developed a method so that we can tolerate noise.
5. For our WCET analysis method, one must choose a family of functions that perform prediction of input to target program to a predicted execution time. There is challenge in how to choose this family. On the one hand, it needs to be very expressive so that it can

describe the reality of the target programs. On the other hand, this expressiveness brings a large number of weights that need to be trained and this requires a large amount of data. We have explored this for affine functions and with affine functions that augmented with other functions that can model interaction effects between bits in the input to the target program. More research is needed on how to choose a family of functions and perform training, and do so in a way that suits the special constraints that we face in our WCET analysis method (ability to extrapolate, having low max magnitude of error) that are different from typical machine learning.

References

Abel, A., & Reineke, J. (2013). Measurement-based modeling of the cache replacement policy. *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS).*

Abella, J., Padilla, M., Castillo, J. D., & Cazorla, F. J. (2017). Measurement-based worst-case execution time estimation using the coefficient of variation. *ACM TODAES*.

Adrian. (2020). *Fixing mass effect black blobs on modern AMD CPUs.* Retrieved from https://cookieplmonster.github.io/2020/07/19/silentpatch-mass-effect/

Altmeyer, S., Lisper, B., Maiza, C., Reineke, J., & Rochange, C. (2015). WCET and mixed-criticality: What does confidence in WCET estimation depend upon? *WCET*.

AMD bulldozer. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Bulldozer_(microarchitecture)

Anantaraman, A., Seth, K., Patil, K., Rotenberg, E., & Mueller, F. (2003). Virtual simple architecture (VISA): Exceeding the complexity limit in safe real-time systems. *ISCA.*

Bartlett, M., Bate, I., & Kazakov, D. (2008). Challenges in relational learning for real-time systems applications. *ILP*.

Bate, I., & Khan, U. (2011). WCET analysis of modern processors using multi-criteria optimization. *Empirical Software Engineering.*

Bernat, G., Colin, A., & Petters, S. M. (2002). WCET analysis of probabilistic hard real-time systems. *RTSS.*

Bernat, G., Colin, A., & Petters, S. M. (2003). *pWCET: A tool for probabilistic worst-case execution time analysis of real-time systems.* Technical Report, YCS-2003-353. University of York.

Betts, A., Merriam, N., & Bernat, G. (2010). Hybrid measurement-based WCET analysis at the source-level using object-level traces. *WCET*.

Bin, J. (2014). Controlling execution time variability using COTS for safety-critical systems. PhD thesis. University Paris Sud.

Bui, T., Mott, M., Vance, W., & Wotell, M. (2020). Insights from preliminary analysis of local cache performance in COTS RTOS for multi-core processors. *DASC.*

Bünte, S., Zolda, M., & Kirner, R. (2011). Let's get less optimistic in measurement-based timing analysis. *IEEE International Symposium on Industrial Embedded Systems.*

Burns, A., & Edgar, S. (2000). Predicting computation time for advanced processor architectures. *ECRTS.*

Cazorla, F. J., Kosmidis, L., Mezzetti, E., Hernandez, C., Abella, J., & Vardanega, T. (2019). Probabilistic worst-case timing analysis: Taxonomy and comprehensive survey. *ACM Comput. Surv*.

Dasgupta, S., Park, D., Kasampalis, T., Adve, V. S., & Roşu, G. (2019). A complete formal semantics of x86-64 user-level instruction set architecture. *PLDI.*

David, L., & Puaut, I. (2004). Static determination of probabilistic execution times. *ECRTS.*

Davis, R. I., Burns, A., Bril, R. J., & Lukkien, J. J. (2007). Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*.

Easdon, C. (2018). *Undocumented CPU behavior: Analyzing undocumented opcodes on Intel x86-64.* Retrieved from IAIK, Graz University of Technology: https://www.cattius.com/images/undocumented-cpu-behavior.pdf

Easdon, C. (2019). *Why audit your CPU? Searching for undocumented CPU behavior.* Retrieved from Third School on Security & Correctness, Graz University of Technology: https://www.cattius.com/images/audit-cpu.pdf

Engblom, J. (2001). On hardware and hardware models for embedded real-time systems. *IEEE RTSS-WIP*.

Ernst, R., & Ye, W. (1997). Embedded program timing analysis based on path clustering and architecture classification. *ICCAD.*

Federal Aviation Administration. (2005). *Real-time scheduling analysis [DOT/FAA/AR-05/27]*.

Federal Aviation Administration. (2006). *Microprocessor evaluations for safety-critical, real-time applications [DOT/FAA/AR-06/34]*.

Ferdinand, C., Heckmann, R., Langenbach, M., Martin, F., Schmidt, M., Theiling, H., . . . Wilhelm, R. (2001, C. Ferdinand; R. Heckmann; M. Langenbach; F. Martin; M. Schmidt; H. Theiling; S. Thesing; & R. Wilhelm). Reliable and precise WCET determination for a real-life processor. *EMSOFT.*

Ferdinand, C., Martin, F., Wilhelm, R., & Alt, M. (1996). Cache behavior prediction by abstract interpretation. *SAS.*

Fog, A. (2015). *4. Instruction tables.* Retrieved from https://www.agner.org/optimize/instruction_tables.pdf

Fog, A. (2015). *Test results for Broadwell and Skylake.* Retrieved from https://www.agner.org/optimize/blog/read.php?i=449#415

Griffin, D., Lesage, B., Bate, I., Soboczenski, F., & Davis, R. I. (2017). Forecast-based interference: modelling multicore interference from observable factors. *RTNS.*

Hansen, J., Hissam, S., & Moreno, G. A. (2009). Statistical-based WCET estimation and validation. *WCET.*

Heule, S., Schkufza, E., Sharma, R., & Aiken, A. (2016). Stratified synthesis: Automatically learning the x86-64 instruction set. *PLDI.*

Holsti, N. (2008). Computing time as a program variable: a way around infeasible paths. *WCET.*

Jacklin, S. (2012). Certification of safety-critical software under DO-178C and DO-278A. *Infotech@ Aerospace*. Retrieved from https://ntrs.nasa.gov/api/citations/20120016835/downloads/20120016835.pdf

Kästner, D., Pister, M., Wegener, S., & Ferdinand, C. (2019). TimeWeaver: A tool for hybrid worst-case execution time analysis. *WCET.*

Kästner, D., Wegener, S., & Ferdinand, C. (2012). Safety standards and WCET analysis tools. *Embedded Real-Time Software and Systems (ERTS).*

Kirner, R., Wenzel, I., Rieder, B., & Puschner, P. P. (2005). Using measurements as a complement to static worst-case execution time analysis. *Intelligence Systems at the Service of Mankind.*

Kosmidis, L., Compagnin, D., Morales, D., Mezzetti, E., Quiñones, E., Abella, J., . . . Cazorla, F. (2016). Measurement-based timing analysis of the AURIX caches. *WCET.*

Kosmidis, L., Quiñones, E., Abella, J., & Vardanega, T. (2016). Fitting processor architectures for measurement-based probabilistic timing analysis. *Microprocessors and Microsystems*.

Lindgren, M. (2000). *Measurement and simulation based techniques for real-time systems analysis.* Licentiate thesis. Uppsala University.

Lundqvist, T., & Stenström, P. (1999). A method to improve the estimated worst-case performance of data caches. *RTCSA.*

Lundqvist, T., & Stenström, P. (1999). An integrated path and timing analysis method based on cycle-level symbolic execution. *Journal of Real-Time Systems*.

Maiza, C., Raymond, P., Parent-Vigouroux, C., Bonenfant, A., Carrier, F., Cassé, H., . . . Sun, W.-T. (2017). The W-SEPT project: Towards semantic-aware WCET estimation. *WCET.*

Mancuso, R., Pellizzoni, R., Caccamo, M., Sha, L., & Yun, H. (2015). WCET(m) estimation in multi-core systems using single core equivalence. *ECRTS.*

Mansur, M. N., Christakis, M., Wüstholz, V., & Zhang, F. (2020). Detecting critical bugs in SMT solvers using blackbox mutational fuzzing. *ESEC/FSE.*

Martins, R. (2022). Towards explainable formal verification. *1st International Workshop on Explainability of Real-time Systems and their Analysis at the IEEE Real-Time Systems Symposium.* Houston, TX. Retrieved from https://sites.google.com/view/ersa22.

Mezzetti, E., & Vardanega, T. (2011). On the industrial fitness of WCET analysis. *WCET.*

Park, J., Winterer, D., Zhang, C., & Su, Z. (2021). Generative type-aware mutation for testing SMT solvers. *OOPSLA.*

Petters, S., & Färber, G. (1999). Making worst case execution time analysis for hard real-time tasks on state of the art processors possible. *RTCSA.*

Petters, S., Zadarnowski, P., & Heiser, G. (2007). Measurements or static analysis or both? *WCET.*

Puschner, P., & Burns, A. (2002). Writing temporally predictable code. *Seventh IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2002).*

Puschner, P., & Koza, C. (1989). Calculating the maximum execution time of real-time programs. *Real-Time Systems*.

Puschner, P., & Nossal, R. (1998). Testing the results of static worst-case execution-time analysis. *IEEE RTSS.*

Quora. (2017). *Which programming language is used to program airplane systems?* Retrieved from https://www.quora.com/Which-programming-language-is-used-to-program-airplane-systems

Rowhammer. (2019). Retrieved from https://en.wikipedia.org/wiki/Row_hammer

Seshia, S. A., & Rakhlin, A. (2008). Game-theoretic timing analysis. *IEEE/ACM International Conference on Computer-Aided Design (ICCAD).*

Sewell, P., Sarkar, S., Nardelli, F. Z., & Myreen, M. O. (2010). x86-TSO: A rigorous and usable programmer's model for x86 multiprocessors. *Communications of the ACM.*

Software considerations in airborne systems and equipment certification [DO-178C]. (2011). RTCA Inc.

Software tool qualification consideration [DO-330]. (2011). RTCA Inc.

Souyris, J., Pavec, E. L., Himbert, G., G. B., V. J., & Heckmann, R. (2005). Computing the worst case execution time of an avionics program by abstract interpretation. *International Workshop on Worst-Case Execution Time (WCET) Analysis.*

Tilbrook, T. M., & McMullen, J. (1990). Washing behind your ears: Principles of software hygiene. *EurOpen.* Retrieved from http://www.qef.com/html/docs/swhygiene.pdf

U.S. Department of Defense. (2014). *Department of Defense handbook---Airworthiness certification criteria. [MIL-HDBK-516C]*.

Vilardell, S., Serra, I., Abella, J., Castillo, J. D., & Cazorla, F. J. (2019). Software timing analysis for complex hardware with survivability and risk analysis. *IEEE 37th International Conference on Computer Design (ICCD).*

Wegener, J., & Grochtmann, M. (1998). Verifying timing constraints of real-time systems by means of evolutionary testing. *Real-Time Systems*.

Wegener, J., & Mueller, F. (2001). A comparison of static analysis and evolutionary testing for the verification of timing constraints. *Journal of Real-Time Systems*.

Wegener, J., Sthamer, H., Jones, B. F., & Eyres, D. E. (1997). Testing real-time systems using genetic algorithms. *Software Quality Journal*.

Wikipedia. (n.d.). *Denormal number.* Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Denormal_number

Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., . . . Stenström, P. (2008). The worst-case execution time problem—Overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS).*

Winterer, D., Zhang, C., & Su, Z. (2020). On the unusual effectiveness of type-aware operator mutations for testing SMT solvers. *OOPSLA.*

Zhang, C., Su, T., Yan, Y., Zhang, F., Pu, G., & Su, Z. (n.d.). Finding and understanding bugs in software model checkers. *ESEC/FSE.* 2019.

Zolda, M., & Kirner, R. (2015). Calculating WCET estimates from timed traces. *Journal of Real-Time Systems*.

A Source code

```
  1 // How to compile:
  2 //   gcc -I/usr/local/include -L/usr/local/lib find_wcet_nn.c -lm -
ltensorflow -pthread -lbsd -march=native -
I/home/ba/gurobi/gurobi952/linux64/include -
L/home/ba/gurobi/gurobi952/linux64/lib -lgurobi95 -o find_wcet_nn
  3
  4 // In order to run it, do the following:
  5 //  cd /home/ba/find_wcet/steepest_ascent_find_wcet/t23
  6 //  sudo -i
  7 //  source /home/ba/.bashrc
  8 //  export GRB_LICENSE_FILE=/root/gurobi.lic
  9 //  in the home directory, there is a directory .local
 10 //    make sure that the the following two directories are there: lib
and bin.
 11 //    if you are useruser and some other user has them, then potentially
set up links to them
 12
 13 #define _GNU_SOURCE
 14 #include <fcntl.h>
 15 #include <stdio.h>
 16 #include <stdlib.h>
 17 #include <string.h>
 18 #include <sys/stat.h>
 19 #include <unistd.h>
 20
 21 #include <math.h>
 22
 23 #include <bsd/stdlib.h>
 24
 25 #include <tensorflow/c/c_api.h>
 26
 27 #include <sched.h>
 28 #include <pthread.h>
 29
 30 #include "gurobi_c.h"
 31
 32 #define PROGRAM_ID_BUBBLESORT                                    0
 33 #define PROGRAM_ID_QUICKSORT                                     1
 34 #define PROGRAM_ID_QSORT                                         2
 35 #define PROGRAM_ID_HEAPSORT                                      3
 36 #define PROGRAM_ID_MERGESORT                                     4
 37 #define PROGRAM_ID_MATVECMUL25                                   5
 38 #define PROGRAM_ID_MATVECMUL700                                  6
 39 #define PROGRAM_ID_FFT_RECT                                      7
 40 #define PROGRAM_ID_FFT_POL                                       8
 41 #define PROGRAM_ID_FFT_REALONLY                                  9
 42 #define PROGRAM_ID_SIMPLEX                                  10
 43 #define inputsizeinnumberofbits                                  32768 //
this must be a multiple of sizeof(int)*8
 44
 45 #define NUMBER_OF_PHYSICAL_PROCESSOR_CORES 32
 46 #define NUM_THREADS_USED_FOR_WORK (NUMBER_OF_PHYSICAL_PROCESSOR_CORES-1)
 47
 48 // #define CLOCK_FREQUENCY_OF_PROCESSOR (3.7*1000000000)
```

```
 49
 50 #define REPLICATION_TO_GET_MEDIAN_OF_EXECUTIONTIME_MEASUREMENTS 5
 51
 52 #define NUMBER_OF_EXAMPLES_FOR_TESTDATASET 32769
 53
 54 #define NUMBER_OF_CONSTRAINTS_FOR_MILP_INITIALIZING_W5_AND_W6 7500000
 55
 56 #define SAMPLING_METHOD_FOR_REGRESSION_WITH_AFFINE_FUNCTION_UNIFORM
0
 57 #define SAMPLING_METHOD_FOR_REGRESSION_WITH_AFFINE_FUNCTION_UNIFORM_TIME
1
 58
 59 #define COMPUTATION_TO_PERFORM_IS_COMPUTE_COUNT_AND_SUM_UNIFORM
0
 60 #define COMPUTATION_TO_PERFORM_IS_COMPUTE_COUNT_AND_SUM_UNIFORM_TIME 1
 61 #define COMPUTATION_TO_PERFORM_IS_UPDATE_PREDICTION_AND_ERROR_SINGLE_BIT
2
 62 #define COMPUTATION_TO_PERFORM_IS_COMPUTE_PREDICTION_AND_ERROR_ALL_BITS
3
 63
 64 #define N_PHASES_ITERATIONS_AFFINE_MODEL 20
 65
 66 int                     ar[inputsizeinnumberofbits/((sizeof(int))*8)]; //
32768 bits = 4096 bytes = 1024 words
 67 int currentinputtoprogram[inputsizeinnumberofbits/((sizeof(int))*8)]; //
this stores an array of bits but it is encoded as an array of int
 68                                                                        //
for example 32768 bits equals 1024 words if one word is 32 bit
 69 void swapdouble(double* p1, double* p2) {
 70   double temp;
 71   temp = *p1;
 72   *p1 = *p2;
 73   *p2 = temp;
 74 }
 75 void swapint(int* p1, int* p2) {
 76   int temp;
 77   temp = *p1;
 78   *p1 = *p2;
 79   *p2 = temp;
 80 }
 81
 82 int sqr(int n) {
 83   return n*n;
 84 }
 85
 86 double sqrdouble(double n) {
 87   return n*n;
 88 }
 89
 90 double mindouble(double t1,double t2) { if (t1<t2) { return t1; } else {
return t2; } }
 91 double maxdouble(double t1,double t2) { if (t1>t2) { return t1; } else {
return t2; } }
 92
 93 __attribute__((optimize("-O3"))) int
getinputtoprogram_size_in_number_of_bytes() {
 94   return inputsizeinnumberofbits/8;
```

```
 95 }
 96
 97 __attribute__((optimize("-O3"))) int
getinputtoprogram_size_in_number_of_ints() {
 98   return inputsizeinnumberofbits/((sizeof(int))*8);
 99 }
100
101 __attribute__((optimize("-O3"))) void clear_int_array(int*
inputtoprogram) {
102   int i;
103   for (i=0;i<getinputtoprogram_size_in_number_of_ints();i++) {
104     inputtoprogram[i] = 0;
105   }
106 }
107
108 __attribute__((optimize("-O3"))) void
setbitzeroin_bitposition_in_integer(int* p_anint, int localbitindex) {
109   int mask;
110   mask = 1 << localbitindex;
111   mask = ~mask;
112   (*p_anint) = (*p_anint) & mask;
113 }
114
115 __attribute__((optimize("-O3"))) void
setbitonein_bitposition_in_integer(int* p_anint, int localbitindex) {
116   int mask;
117   mask = 1 << localbitindex;
118   (*p_anint) = (*p_anint) | mask;
119 }
120
121 __attribute__((optimize("-O3"))) int obtain_intindex_from_bitindex(int
bitindex) {
122   int intindex;
123   intindex = bitindex / ((sizeof(int))*8);
124   return intindex;
125 }
126 __attribute__((optimize("-O3"))) int
obtain_localbitindex_from_bitindex(int bitindex) {
127   int localbitindex;
128   localbitindex = bitindex % ((sizeof(int))*8);
129   return localbitindex;
130 }
131
132 __attribute__((optimize("-O3"))) void
setbitzeroin_bitposition_in_integerarray(int* p_anintarray, int bitindex) {
133   int intindex; int localbitindex;
134   intindex      = obtain_intindex_from_bitindex(      bitindex);
135   localbitindex = obtain_localbitindex_from_bitindex( bitindex);
136   setbitzeroin_bitposition_in_integer(&(p_anintarray[intindex]),
localbitindex);
137 }
138
139 __attribute__((optimize("-O3"))) void
setbitonein_bitposition_in_integerarray(int* p_anintarray, int bitindex) {
140   int intindex; int localbitindex;
141   intindex      = obtain_intindex_from_bitindex(      bitindex);
142   localbitindex = obtain_localbitindex_from_bitindex( bitindex);
```

```
143   setbitonein_bitposition_in_integer(&(p_anintarray[intindex]),
localbitindex);
144 }
145
146 __attribute__((optimize("-O3"))) int
isbitonein_bitposition_in_integerarray(int* p_anintarray, int bitindex) {
147   if ((p_anintarray[bitindex/((sizeof(int))*8)]) & (1 <<
(bitindex%((sizeof(int))*8)))) {
148     return 1;
149   } else {
150     return 0;
151   }
152 }
153
154 __attribute__((optimize("-O3"))) void
fill_int_array_with_random_bits(int* inputtoprogram) {
155   int i; int localbitindex;
156   clear_int_array(inputtoprogram);
157   for (i=0;i<getinputtoprogram_size_in_number_of_ints();i++) {
158     for
(localbitindex=0;localbitindex<((sizeof(int))*8);localbitindex++) {
159       if (drand48()<=0.5) {
160         setbitonein_bitposition_in_integer(&(inputtoprogram[i]),
localbitindex);
161       }
162     }
163   }
164 }
165
166 __attribute__((optimize("-O3"))) void copy_inputtoprogram(int*
inputtoprogram_destination, int* inputtoprogram_source) {
167   int i;
168   for (i=0;i<getinputtoprogram_size_in_number_of_ints();i++) {
169     inputtoprogram_destination[i] = inputtoprogram_source[i];
170   }
171 }
172
173 // *** BEGIN: Here comes code that is used for running target programs
174
175 double compute_square(double x) {
176   return x*x;
177 }
178
179 unsigned long long diff_ns_timespec(struct timespec* x, struct timespec*
y) {
180   unsigned long long abillion = 1000000000;
181   unsigned long long u;
182   u = x->tv_sec;
183   u = u * abillion;
184   u = u + x->tv_nsec;
185   unsigned long long v;
186   v = y->tv_sec;
187   v = v * abillion;
188   v = v + y->tv_nsec;
189   return u-v;
190 }
191 double diff_s_timespec(struct timespec* x, struct timespec* y) {
```

```
192   unsigned long long abillion = 1000000000;
193   unsigned long long t;
194   unsigned long long tsec;
195   unsigned long long tnsec;
196   double t_as_double_secondpart;
197   double t_as_double_nsecondpart;
198   double t_as_double;
199   t = diff_ns_timespec( x, y);
200   tsec = t / abillion;
201   tnsec = t % abillion;
202   t_as_double_secondpart  = tsec;
203   t_as_double_nsecondpart = tnsec;
204   t_as_double_nsecondpart = t_as_double_nsecondpart / abillion;
205   t_as_double = t_as_double_secondpart + t_as_double_nsecondpart;
206   if (t_as_double<0) { printf("Error diff_s_timespec.
t_as_double=%lf\n", t_as_double); }
207   return t_as_double;
208 }
209
210 void swap(int* p1, int *p2) {
211   int temp;
212   temp = *p1;
213   *p1 = *p2;
214   *p2 = temp;
215 }
216
217 void bubblesort(int n) {
218   int i; int swapped;
219   do {
220     swapped = 0;
221     for (i=1;i<=n-1;i++) {
222       if (ar[i-1] > ar[i]) {
223         swap( &(ar[i-1]), &(ar[i]) );
224         swapped = 1;
225       }
226     }
227   } while (swapped);
228 }
229
230 void quicksort(int first, int last) {
231   int i; int j; int pivot; int temp;
232   if (first<last) {
233     pivot = first;
234     i     = first;
235     j     = last;
236     while(i<j) {
237       while ((ar[i]<=ar[pivot]) && (i<last)) {
238         i++;
239       }
240       while (ar[j]>ar[pivot]) {
241         j--;
242       }
243       if(i<j) {
244         temp=ar[i];
245         ar[i]=ar[j];
246         ar[j]=temp;
247       }
```

```
 248     }
 249     temp=ar[pivot];
 250     ar[pivot]=ar[j];
 251     ar[j]=temp;
 252     quicksort(first,j-1);
 253     quicksort(j+1,last);
 254   }
 255 }
 256
 257 #define INPUTSIZE_IN_TERMS_OF_NUMBER_OF_UINT32
(inputsizeinnumberofbits/((sizeof(int))*8))
 258 #define INPUTSIZE_IN_TERMS_OF_NUMBER_OF_DOUBLE
(INPUTSIZE_IN_TERMS_OF_NUMBER_OF_UINT32/2)
 259 double Amatrix25[  25*INPUTSIZE_IN_TERMS_OF_NUMBER_OF_DOUBLE];  double
uvector25[25];
 260 double Amatrix700[700*INPUTSIZE_IN_TERMS_OF_NUMBER_OF_DOUBLE];  double
uvector700[700];
 261 double getmatrixelement25( int row,int col)          { return Amatrix25[
row*INPUTSIZE_IN_TERMS_OF_NUMBER_OF_DOUBLE+col];  }
 262 double getmatrixelement700(int row,int col)          { return
Amatrix700[row*INPUTSIZE_IN_TERMS_OF_NUMBER_OF_DOUBLE+col];  }
 263 void setmatrixelement25( int row,int col,double v) { Amatrix25[
row*INPUTSIZE_IN_TERMS_OF_NUMBER_OF_DOUBLE+col] = v;    }
 264 void setmatrixelement700(int row,int col,double v) { Amatrix700[
row*INPUTSIZE_IN_TERMS_OF_NUMBER_OF_DOUBLE+col] = v;    }
 265 void write_Amatrix_to_file25(char* fn) {
 266   int row; int col; FILE* f;
 267   f = fopen(fn, "w+" );
 268   for (row=0;row<25;row++) {
 269     for (col=0;col<INPUTSIZE_IN_TERMS_OF_NUMBER_OF_DOUBLE;col++) {
 270       fprintf(f,"%d %d %lf\n", row, col, getmatrixelement25(row,col));
 271     }
 272   }
 273   fclose(f);
 274 }
 275 void write_Amatrix_to_file700(char* fn) {
 276   int row; int col; FILE* f;
 277   f = fopen(fn, "w+" );
 278   for (row=0;row<700;row++) {
 279     for (col=0;col<INPUTSIZE_IN_TERMS_OF_NUMBER_OF_DOUBLE;col++) {
 280       fprintf(f,"%d %d %lf\n", row, col, getmatrixelement700(row,col));
 281     }
 282   }
 283   fclose(f);
 284 }
 285 void initialize_Amatrix25() {
 286   int row; int col;
 287   char fn[2000];
 288   for (row=0;row<25;row++) {
 289     for (col=0;col<INPUTSIZE_IN_TERMS_OF_NUMBER_OF_DOUBLE;col++) {
 290       setmatrixelement25(row,col,drand48());
 291     }
 292   }
 293   sprintf(fn,"Amatrix25.txt");
 294   // write_Amatrix_to_file25("Amatrix25.txt");
 295   write_Amatrix_to_file25(fn);
 296 }
```

```
297 void initialize_Amatrix700() {
298   int row; int col;
299   char fn[2000];
300   for (row=0;row<700;row++) {
301     for (col=0;col<INPUTSIZE_IN_TERMS_OF_NUMBER_OF_DOUBLE;col++) {
302       setmatrixelement700(row,col,drand48());
303     }
304   }
305   sprintf(fn,"Amatrix700.txt");
306   write_Amatrix_to_file700(fn);
307 }
308 void matvecmul25() {
309   int row; int col;
310   double* p_double_array;
311   p_double_array = ((double*) ar);
312   for (row=0;row<25;row++) {
313     uvector25[row] = 0.0;
314     for (col=0;col<INPUTSIZE_IN_TERMS_OF_NUMBER_OF_DOUBLE;col++) {
315       uvector25[row] = uvector25[row] + getmatrixelement25(row,col) *
p_double_array[col];
316     }
317   }
318 }
319 void matvecmul700() {
320   int row; int col;
321   double* p_double_array;
322   p_double_array = ((double*) ar);
323   for (row=0;row<700;row++) {
324     uvector700[row] = 0.0;
325     for (col=0;col<INPUTSIZE_IN_TERMS_OF_NUMBER_OF_DOUBLE;col++) {
326       uvector700[row] = uvector700[row] + getmatrixelement700(row,col) *
p_double_array[col];
327     }
328   }
329 }
330
331 struct complex_double {
332   double real_part;
333   double imag_part;
334 };
335
336 struct complex_double create_complex_double_rectangular_form(double
real_part,double imag_part) {
337   struct complex_double temp;
338   temp.real_part = real_part;
339   temp.imag_part = imag_part;
340   return temp;
341 }
342
343 struct complex_double create_complex_double_polar_form(double
absolute_part,double argument_part) {
344   struct complex_double temp;
345   temp.real_part = absolute_part * cos(argument_part);
346   temp.imag_part = absolute_part * sin(argument_part);
347   return temp;
348 }
349
```

```
350 struct complex_double complex_double_multiply(struct complex_double
a_complex_number1,struct complex_double a_complex_number2) {
351   double real_part; double imag_part;
352   real_part = a_complex_number1.real_part * a_complex_number2.real_part
- a_complex_number1.imag_part * a_complex_number2.imag_part;
353   imag_part = a_complex_number1.real_part * a_complex_number2.imag_part
+ a_complex_number1.imag_part * a_complex_number2.real_part;
354   return create_complex_double_rectangular_form(real_part,imag_part);
355 }
356
357 struct complex_double complex_double_add(struct complex_double
a_complex_number1,struct complex_double a_complex_number2) {
358   double real_part; double imag_part;
359   real_part = a_complex_number1.real_part + a_complex_number2.real_part;
360   imag_part = a_complex_number1.imag_part + a_complex_number2.imag_part;
361   return create_complex_double_rectangular_form(real_part,imag_part);
362 }
363
364 struct complex_double complex_double_subtract(struct complex_double
a_complex_number1,struct complex_double a_complex_number2) {
365   double real_part; double imag_part;
366   real_part = a_complex_number1.real_part - a_complex_number2.real_part;
367   imag_part = a_complex_number1.imag_part - a_complex_number2.imag_part;
368   return create_complex_double_rectangular_form(real_part,imag_part);
369 }
370
371 // this code assumes that the_exponent>=0
372 struct complex_double complex_double_power(struct complex_double
the_complex_number,int the_exponent) {
373   struct complex_double temp_complex_number;
374   if (the_exponent==0) {
375     return create_complex_double_rectangular_form(1.0,0.0);
376   } else {
377     if (the_exponent==1) {
378       return the_complex_number;
379     } else {
380       if (the_exponent%2==1) {
381         temp_complex_number =
complex_double_power(the_complex_number,the_exponent-1);
382         return
complex_double_multiply(temp_complex_number,the_complex_number);
383       } else {
384         temp_complex_number =
complex_double_power(the_complex_number,the_exponent/2);
385         return
complex_double_multiply(temp_complex_number,temp_complex_number);
386       }
387     }
388   }
389 }
390
391 struct complex_double vec[
(inputsizeinnumberofbits/(8*(sizeof(int))))/2];
392 struct complex_double vec2[(inputsizeinnumberofbits/(8*sizeof(int)))/2];
393 struct complex_double W[((inputsizeinnumberofbits/(8*sizeof(int)))/2)];
394
```

```
// it is assumed that N is positive. Specifically, if N is zero, then this code will not work. Also, if N is negative, then the shifting below might be problematic
int mylog2(int N) {
  int k; int i;
  k = N; i = 0;
  while(k) {
    k = k >> 1;
    i++;
  }
  return i - 1;
}

int reverse(int N, int n) {
  int j; int p;
  p = 0;
  for (j=1;j<=mylog2(N);j++) {
    if (n & (1 << (mylog2(N) - j))) {
      p = p | (1 << (j - 1));
    }
  }
  return p;
}

void change_order(int N) {
  for(int i = 0; i < N; i++) {
    vec2[i] = vec[reverse( N, i)];
  }
  for(int j = 0; j < N; j++) {
    vec[j] = vec2[j];
  }
}

void do_fft_transform(int N) {
  int n; int a; int j; int i;
  struct complex_double temp1;
  struct complex_double temp2;
  change_order(N);
  W[1] = create_complex_double_polar_form( 1.0, -2.0 * M_PI / N );
  W[0] = create_complex_double_polar_form( 1.0, 0.0             );
  for (i=2;i<N/2;i++) {
    W[i] = complex_double_power(W[1], i);
  }
  n = 1;
  a = N / 2;
  for (j=0;j<mylog2(N);j++) {
    for (i=0;i<N;i++) {
      if (!(i & n)) {
        temp1    = vec[i];
        temp2    = complex_double_multiply(W[(i * a) % (n * a)],vec[i + n]);

        vec[i]   = complex_double_add(     temp1,temp2);
        vec[i+n] = complex_double_subtract(temp1,temp2);
      }
    }
    n = n * 2;
    a = a / 2;
```

```
 449   }
 450 }
 451
 452 void fft_rect() {
 453   int N; int i; double v1; double v2;
 454   N = (inputsizeinnumberofbits/(8*sizeof(int)))/4;
 455   for (i=0;i<N;i++) {
 456     v1 = (*  ((double*) (&(ar[i  ]))) );
 457     v2 = (*  ((double*) (&(ar[i+2]))) );
 458     vec[i] = create_complex_double_rectangular_form(v1,v2);
 459   }
 460   do_fft_transform(N);
 461 }
 462
 463 void fft_pol() {
 464   int N; int i; double v1; double v2;
 465   N = (inputsizeinnumberofbits/(8*sizeof(int)))/4;
 466   for (i=0;i<N;i++) {
 467     v1 = (*  ((double*) (&(ar[i  ]))) );
 468     v2 = (*  ((double*) (&(ar[i+2]))) );
 469     vec[i] = create_complex_double_polar_form(v1,v2);
 470   }
 471   do_fft_transform(N);
 472 }
 473
 474 void fft_realonly() {
 475   int N; int i; double v1;
 476   N = (inputsizeinnumberofbits/(8*sizeof(int)))/2; // note that here we
divide by 2 rather than 4
 477   for (i=0;i<N;i++) {
 478     v1 = (*  ((double*) (&(ar[i  ]))) );
 479     vec[i] = create_complex_double_rectangular_form(v1,0.0);
 480   }
 481   do_fft_transform(N);
 482 }
 483
 484 #define MAXARSIZEFORSIMPLEX 2048
 485
 486 double EPSILON_USED_FOR_SIMPLEX = 1.0E-10;
 487
 488 int arsize;
 489 int dimension_of_A_matrix;
 490
 491 double Ainp[MAXARSIZEFORSIMPLEX];
 492 double binp[MAXARSIZEFORSIMPLEX];
 493 double cinp[MAXARSIZEFORSIMPLEX];
 494
 495 int m;
 496 int n;
 497 double a[MAXARSIZEFORSIMPLEX];  // this array needs to have enough space
for (m + 1)*(n + m + 1) double
 498
 499 int basis[MAXARSIZEFORSIMPLEX]; // this needs to be large enough for m
integers
 500
 501 double x[MAXARSIZEFORSIMPLEX];  // this will be used to store primal of
solution; n elements
```

```
502 double y[MAXARSIZEFORSIMPLEX];  // this will be used to store dual   of
solution; m elements
503
504 int statusflag_LP_is_unbounded;
505 int statusflag_termminated_without_optimal_feasible_solution;
506
507 double myfabs(double t0) {
508   if (t0>=0.0) {
509     return t0;
510   } else {
511     return (-1.0)*t0;
512   }
513 }
514
515 double get_ci_element_in_ar(int i) {
516   return ((double*) ar)[i];
517 }
518 double get_bi_element_in_ar(int i) {
519   return ((double*) ar)[dimension_of_A_matrix+i];
520 }
521 double get_Aij_element_in_ar(int i, int j) {
522   return ((double*)
ar)[2*dimension_of_A_matrix+i*dimension_of_A_matrix+j];
523 }
524 double get_element_in_cinp(int i) {
525   return cinp[i];
526 }
527 void set_element_in_cinp(int i,double v) {
528   cinp[i] = v;
529 }
530 double get_element_in_binp(int i) {
531   return binp[i];
532 }
533 void set_element_in_binp(int i,double v) {
534   binp[i] = v;
535 }
536 int get_index_for_Ainp(int i,int j) {
537   return i*dimension_of_A_matrix+j;
538 }
539 double get_element_in_Ainp(int i,int j) {
540   int index_for_Ainp;
541   index_for_Ainp = get_index_for_Ainp(i,j);
542   return Ainp[index_for_Ainp];
543 }
544 void set_element_in_Ainp(int i,int j,double v) {
545   int index_for_Ainp;
546   index_for_Ainp = get_index_for_Ainp(i,j);
547   Ainp[index_for_Ainp] = v;
548 }
549 int get_index_for_a(int i,int j) {
550   return i*(n+m+1)+j;
551 }
552 double get_element_in_a(int i,int j) {
553   int index_for_a;
554   index_for_a = get_index_for_a(i,j);
555   return a[index_for_a];
556 }
```

```
557 void set_element_in_a(int i,int j,double v) {
558   int index_for_a;
559   index_for_a = get_index_for_a(i,j);
560   a[index_for_a] = v;
561 }
562
563 // This assumes that 3<=x because then a lower bound on the sought after
quantity is 1 and an upper bound is t0.
564 // We implement it this way because in this way, we don't need to take
square root and hence we do not depend on math library.
565 int compute_largest_x_such_that_xsquared_plus_2x_is_atmost(int t0) {
566   int lb; int ub; int mid;
567   lb = 1; ub = t0;
568   while (lb<ub) {
569     mid = (lb+ub)/2;
570     if (mid*mid+2*mid==t0) {
571       return mid;
572     } else {
573       if (mid*mid+2*mid<t0) {
574         if ((mid+1)*(mid+1)+2*(mid+1)>t0) {
575           return mid;
576         } else {
577           lb = mid+1;
578         }
579       } else {
580         if ((mid-1)*(mid-1)+2*(mid-1)<=t0) {
581           return mid-1;
582         } else {
583           ub = mid-2;
584         }
585       }
586     }
587   }
588   printf("Error in
compute_largest_x_such_that_xsquared_plus_2x_is_atmost.\n"); exit(-1);
589 }
590
591 void fill_Ainp_binp_cinp_from_ar() {
592   int i; int j; double v;
593   for (i=0;i<dimension_of_A_matrix;i++) {
594     v = get_ci_element_in_ar(i);
595     set_element_in_cinp(i,v);
596   }
597   for (i=0;i<dimension_of_A_matrix;i++) {
598     v = get_bi_element_in_ar(i);
599     set_element_in_binp(i,v);
600   }
601   for (i=0;i<dimension_of_A_matrix;i++) {
602     for (j=0;j<dimension_of_A_matrix;j++) {
603       v = get_Aij_element_in_ar(i,j);
604       set_element_in_Ainp(i,j,v);
605     }
606   }
607 }
608 int bland() {
609   int j;
610   for (j=0;j<m+n;j++) {
```

```
611     if (get_element_in_a(m,j)>0) {
612       return j;
613     }
614   }
615   return -1;
616 }
617
618 int minRatioRule(int q) {
619   int i; int p;
620   p = -1;
621   for (i=0;i<m;i++) {
622     if (get_element_in_a(i,q) > EPSILON_USED_FOR_SIMPLEX) {
623       if (p==-1) {
624         p = i;
625       } else if ((get_element_in_a(i,m+n)/get_element_in_a(i,q)) <
(get_element_in_a(p,m+n)/get_element_in_a(p,q))) {
626         p = i;
627       }
628     }
629   }
630   return p;
631 }
632
633 void pivot(int p, int q) {
634   int i; int j;
635   for (i=0;i<=m;i++) {
636     for (j=0;j<=m+n;j++) {
637       if ((i!=p) && (j!=q)) {
638         set_element_in_a(i,j,   get_element_in_a(i,j) -
get_element_in_a(p,j) * get_element_in_a(i,q) / get_element_in_a(p,q)   );
639       }
640     }
641   }
642   for (i=0;i<=m;i++) {
643     if (i!=p) {
644       set_element_in_a(i,q,0.0);
645     }
646   }
647   for (j=0;j<=m+n;j++) {
648     if (j!=q) {
649       set_element_in_a(p,j,
get_element_in_a(p,j)/get_element_in_a(p,q)   );
650     }
651   }
652   set_element_in_a(p,q,1.0);
653 }
654
655 void solve() {
656   int q; int p;
657   q = bland();
658   while (q!=-1) {
659     p = minRatioRule(q);
660     if (p==-1) {
661       statusflag_LP_is_unbounded = 1;
662       return;
663     }
664     pivot(p,q);
```

```
665       basis[p] = q;
666       q = bland();
667     }
668   }
669
670   double get_value_of_objective_function() {
671     return -get_element_in_a(m,m+n);
672   }
673
674   void fill_primal() {
675     int i;
676     for (i=0;i<m;i++) {
677       if (basis[i]<n) {
678         x[basis[i]] = get_element_in_a(i,m+n);
679       }
680     }
681   }
682
683   void fill_dual() {
684     int i;
685     for (i=0;i<m;i++) {
686       if (basis[i]<n) {
687         y[i] = (-1)*get_element_in_a(m,n+i);
688       }
689     }
690   }
691
692   int isprimalfeasible() {
693     int i; int j; double sum;
694     fill_primal();
695     for (j=0;j<n;j++) {
696       if (x[j]<0.0) {
697         return 0;
698       }
699     }
700     for (i=0;i<m;i++) {
701       sum = 0.0;
702       for (j=0;j<n;j++) {
703         sum = sum + get_element_in_Ainp(i,j) * x[j];
704       }
705       if (sum > get_element_in_binp(i)+EPSILON_USED_FOR_SIMPLEX) {
706         return 0;
707       }
708     }
709     return 1;
710   }
711
712   int isdualfeasible() {
713     int i; int j; double sum;
714     fill_dual();
715     for (i=0;i<m;i++) {
716       if (y[i]<0.0) {
717         return 0;
718       }
719     }
720     for (j=0;j<n;j++) {
721       sum = 0.0;
```

```
722       for (i=0;i<m;i++) {
723         sum = sum + get_element_in_Ainp(i,j) * y[i];
724       }
725       if (sum<get_element_in_cinp(j)-EPSILON_USED_FOR_SIMPLEX) {
726         return 0;
727       }
728     }
729     return 1;
730   }
731
732   int isoptimal() {
733     int i; int j; double value; double value1; double value2;
734     fill_primal();
735     fill_dual();
736     value = get_value_of_objective_function();
737     value1 = 0.0;
738     for (j=0;j<n;j++) {
739       value1 = value1 + get_element_in_cinp(j) * x[j];
740     }
741     value2 = 0.0;
742     for (i=0;i<m;i++) {
743       value2 = value2 + y[i] * get_element_in_binp(i);
744     }
745     if (myfabs(value - value1) > EPSILON_USED_FOR_SIMPLEX || myfabs(value
- value2) > EPSILON_USED_FOR_SIMPLEX) {
746       return 0;
747     }
748     return 1;
749   }
750
751   int check_solution_of_simplex() {
752     return isprimalfeasible() && isdualfeasible() && isoptimal();
753   }
754
755   void clear_a_matrix() {
756     int i; int j;
757     for (i=0;i<m+1;i++) {
758       for (j=0;j<n+m+1;j++) {
759         set_element_in_a(i,j,0.0);
760       }
761     }
762   }
763
764   void simplex() {
765     int i; int j; double value_of_objective_function;
766     arsize = 512;
767     if (arsize>=3) { // if this is false, then the input array is too
small so we can't compute dimension_of_A_matrix
768       dimension_of_A_matrix =
compute_largest_x_such_that_xsquared_plus_2x_is_atmost(arsize);
769       fill_Ainp_binp_cinp_from_ar();
770       m = dimension_of_A_matrix;
771       n = dimension_of_A_matrix;
772       for (i=0;i<m;i++) {
773         if (!(binp[i] >= 0)) {
774           binp[i] = -binp[i]; // we do not allow negative rhs
775         }
```

```
776     }
777     clear_a_matrix();
778     for (i=0;i<m;i++) {
779       for (j=0;j<n;j++) {
780         set_element_in_a(i,j,get_element_in_Ainp(i,j));
781       }
782     }
783     for (i=0;i<m;i++) {
784       set_element_in_a(i,n+i,1.0);
785     }
786     for (j=0;j<n;j++) {
787       set_element_in_a(m,j,get_element_in_cinp(j));
788     }
789     for (i=0;i<m;i++) {
790       set_element_in_a(i,m+n,get_element_in_binp(i));
791     }
792     for (i=0;i<m;i++) {
793       basis[i] = n + i;
794     }
795     statusflag_LP_is_unbounded = 0;
796     statusflag_termminated_without_optimal_feasible_solution = 0;
797     solve();
798     if (!check_solution_of_simplex()) {
799       statusflag_termminated_without_optimal_feasible_solution = 1;
800     } else {
801       value_of_objective_function = get_value_of_objective_function();
802     }
803   }
804 }
805
806 int cmpfunc (const void * a, const void * b) {
807 // Some examples on the web show the following code
808 //   return ( *(int*)a - *(int*)b );
809 // Unfortunately, that does not work because it leads to overflow
810 // Therefore, we use the code below instead.
811   if ( *((int*)a) < *((int*)b)) {
812     return -1;
813   } else {
814     if ( *((int*)a) == (*(int*)b)) {
815       return 0;
816     } else {
817       return 1;
818     }
819   }
820 }
821
822 int cmpfuncdouble( const void * a, const void * b) {
823   if ( *((double*)a) < *((double*)b)) {
824     return -1;
825   } else {
826     if ( *((double*)a) == (*(double*)b)) {
827       return 0;
828     } else {
829       return 1;
830     }
831   }
832 }
```

```
 833
 834 // We want to measure the time it takes to run a program. To do this, we
read the current time, then run the program, then read the current time, then
take the difference
 835 // between these two times. There are two different ways to get the
current time: read time stamp count using an instruction or make the call
clock_gettime.
 836 // We have used both and there seems to be no major difference between
them. We started out with clock_gettime and when were worried that it may
make a system call
 837 // which invokes the operating system and then executes other functions
(slow interrupts) and this would disturb the timing measurements. Therefore,
 838 // we thought that rdtsc might be better. But we did not see any big
difference. Earlier in our investigation, we also found some rare cases (one
in 10 million) when
 839 // the executiontime get 0.8ms longer; we thought this may have been
triggered somehow by the clock_gettime; but it turns out that we experienced
the same thing with rdtsc.
 840 // Since we get approximately the same behavior with clock_gettime and
rdtsc, we use clock_gettime because this is more portable.
 841
 842 // void rdtscl(unsigned long long *ll)
 843 // {
 844 //     unsigned int lo, hi;
 845 //     __asm__ __volatile__ ("lfence;rdtsc;lfence" : "=a"(lo),
"=d"(hi));
 846 //     *ll = ( (unsigned long long)lo)|( ((unsigned long long)hi)<<32 );
 847 // }
 848
 849 double run_program_with_input_dont_use_filesystem(int* inputtoprogram,
int programid) {
 850   int status_clock_gettime_begin;
 851   int status_clock_gettime_end;
 852   // unsigned long long mybegin;
 853   // unsigned long long myend;
 854   // unsigned long long mydifference;
 855   struct timespec mybegin;
 856   struct timespec myend;
 857   double t;
 858   copy_inputtoprogram(ar, inputtoprogram);
 859   status_clock_gettime_begin = clock_gettime(CLOCK_MONOTONIC_RAW,
&mybegin);
 860   if (status_clock_gettime_begin!=0) { printf("Error in
run_program_with_input_dont_use_filesystem.
status_clock_gettime_begin=%d\n",status_clock_gettime_begin); exit(-1); }
 861   // rdtscl(&mybegin);
 862   if (programid==PROGRAM_ID_BUBBLESORT) {
 863     bubblesort(getinputtoprogram_size_in_number_of_ints());
 864   } else {
 865     if (programid==PROGRAM_ID_QUICKSORT) {
 866       quicksort(0, getinputtoprogram_size_in_number_of_ints()-1);
 867     } else {
 868       if (programid==PROGRAM_ID_QSORT) {
 869         qsort(ar, getinputtoprogram_size_in_number_of_ints(),
sizeof(int), cmpfunc);
 870       } else {
 871         if (programid==PROGRAM_ID_HEAPSORT) {
```

```
872           heapsort(ar, getinputtoprogram_size_in_number_of_ints(),
sizeof(int), cmpfunc);
873         } else {
874           if (programid==PROGRAM_ID_MERGESORT) {
875             mergesort(ar, getinputtoprogram_size_in_number_of_ints(),
sizeof(int), cmpfunc);
876           } else {
877             if (programid==PROGRAM_ID_MATVECMUL25) {
878               matvecmul25();
879             } else {
880               if (programid==PROGRAM_ID_MATVECMUL700) {
881                 matvecmul700();
882               } else {
883                 if (programid==PROGRAM_ID_FFT_RECT) {
884                   fft_rect();
885                 } else {
886                   if (programid==PROGRAM_ID_FFT_POL) {
887                     fft_pol();
888                   } else {
889                     if (programid==PROGRAM_ID_FFT_REALONLY) {
890                       fft_realonly();
891                     } else {
892                       if (programid==PROGRAM_ID_SIMPLEX) {
893                         simplex();
894                       } else {
895                         printf("Error. Program is not supported.\n");
896                         exit(-1);
897                       }
898                     }
899                   }
900                 }
901               }
902             }
903           }
904         }
905       }
906     }
907   }
908   status_clock_gettime_end = clock_gettime(CLOCK_MONOTONIC_RAW, &myend);
909   if (status_clock_gettime_end!=0) { printf("Error in
run_program_with_input_dont_use_filesystem.
status_clock_gettime_end=%d\n",status_clock_gettime_end); exit(-1); }
910   // rdtscl(&myend);
911   t = diff_s_timespec(&myend, &mybegin);
912   // mydifference = myend-mybegin;
913   // t = mydifference;
914   // t = t/CLOCK_FREQUENCY_OF_PROCESSOR;
915   // printf("Done execution. t = %lf\n",t); fflush(stdout);
916   return t;
917 }
918
919 void sort_recorded_measured_execution_times_ascending_order(double*
recorded_measured_execution_times,int n_elements) {
920   int iterator1; int iterator2;
921   for (iterator1=0;iterator1<n_elements;iterator1++) {
922     for (iterator2=iterator1+1;iterator2<n_elements;iterator2++) {
```

```
 923       if
(recorded_measured_execution_times[iterator1]>recorded_measured_execution_tim
es[iterator2]) {
 924
swapdouble(&(recorded_measured_execution_times[iterator1]),&(recorded_measure
d_execution_times[iterator2]));
 925       }
 926     }
 927   }
 928 }
 929
 930 void
run_program_with_three_different_inputs_dont_use_filesystem_run_X_times(int*
inputtoprogram0,int* inputtoprogram1,int* inputtoprogram2,int programid,int
ntimes,double* p_t0,double* p_t1,double* p_t2) {
 931   double* recorded_measured_execution_times0;
 932   double* recorded_measured_execution_times1;
 933   double* recorded_measured_execution_times2;
 934   int iterator;
 935   recorded_measured_execution_times0 = (double*) malloc( sizeof(double)
* ntimes );
 936   recorded_measured_execution_times1 = (double*) malloc( sizeof(double)
* ntimes );
 937   recorded_measured_execution_times2 = (double*) malloc( sizeof(double)
* ntimes );
 938   recorded_measured_execution_times0[0] =
run_program_with_input_dont_use_filesystem(inputtoprogram0,programid);
 939   recorded_measured_execution_times0[0] =
run_program_with_input_dont_use_filesystem(inputtoprogram0,programid);
 940   recorded_measured_execution_times1[0] =
run_program_with_input_dont_use_filesystem(inputtoprogram1,programid);
 941   recorded_measured_execution_times1[0] =
run_program_with_input_dont_use_filesystem(inputtoprogram1,programid);
 942   recorded_measured_execution_times2[0] =
run_program_with_input_dont_use_filesystem(inputtoprogram2,programid);
 943   recorded_measured_execution_times2[0] =
run_program_with_input_dont_use_filesystem(inputtoprogram2,programid);
 944   recorded_measured_execution_times0[0] =
run_program_with_input_dont_use_filesystem(inputtoprogram0,programid);
 945   recorded_measured_execution_times0[0] =
run_program_with_input_dont_use_filesystem(inputtoprogram0,programid);
 946   recorded_measured_execution_times1[0] =
run_program_with_input_dont_use_filesystem(inputtoprogram1,programid);
 947   recorded_measured_execution_times1[0] =
run_program_with_input_dont_use_filesystem(inputtoprogram1,programid);
 948   recorded_measured_execution_times2[0] =
run_program_with_input_dont_use_filesystem(inputtoprogram2,programid);
 949   recorded_measured_execution_times2[0] =
run_program_with_input_dont_use_filesystem(inputtoprogram2,programid);
 950   for (iterator=0;iterator<ntimes;iterator++) {
 951     recorded_measured_execution_times0[iterator] =
run_program_with_input_dont_use_filesystem(inputtoprogram0,programid);
 952     recorded_measured_execution_times0[iterator] =
run_program_with_input_dont_use_filesystem(inputtoprogram0,programid);
 953     recorded_measured_execution_times1[iterator] =
run_program_with_input_dont_use_filesystem(inputtoprogram1,programid);
```

```
 954     recorded_measured_execution_times1[iterator] =
run_program_with_input_dont_use_filesystem(inputtoprogram1,programid);
 955     recorded_measured_execution_times2[iterator] =
run_program_with_input_dont_use_filesystem(inputtoprogram2,programid);
 956     recorded_measured_execution_times2[iterator] =
run_program_with_input_dont_use_filesystem(inputtoprogram2,programid);
 957   }
 958
sort_recorded_measured_execution_times_ascending_order(recorded_measured_exec
ution_times0,ntimes);
 959
sort_recorded_measured_execution_times_ascending_order(recorded_measured_exec
ution_times1,ntimes);
 960
sort_recorded_measured_execution_times_ascending_order(recorded_measured_exec
ution_times2,ntimes);
 961   *p_t0 = recorded_measured_execution_times0[ntimes/2];
 962   *p_t1 = recorded_measured_execution_times1[ntimes/2];
 963   *p_t2 = recorded_measured_execution_times2[ntimes/2];
 964   free(recorded_measured_execution_times0);
 965   free(recorded_measured_execution_times1);
 966   free(recorded_measured_execution_times2);
 967 }
 968
 969 void
run_program_with_two_different_inputs_dont_use_filesystem_run_X_times(int*
inputtoprogram0,int* inputtoprogram1,int programid,int ntimes,double*
p_t0,double* p_t1) {
 970   double* recorded_measured_execution_times0;
 971   double* recorded_measured_execution_times1;
 972   int iterator;
 973   recorded_measured_execution_times0 = (double*) malloc( sizeof(double)
* ntimes );
 974   recorded_measured_execution_times1 = (double*) malloc( sizeof(double)
* ntimes );
 975   recorded_measured_execution_times0[0] =
run_program_with_input_dont_use_filesystem(inputtoprogram0,programid);
 976   recorded_measured_execution_times0[0] =
run_program_with_input_dont_use_filesystem(inputtoprogram0,programid);
 977   recorded_measured_execution_times1[0] =
run_program_with_input_dont_use_filesystem(inputtoprogram1,programid);
 978   recorded_measured_execution_times1[0] =
run_program_with_input_dont_use_filesystem(inputtoprogram1,programid);
 979   recorded_measured_execution_times0[0] =
run_program_with_input_dont_use_filesystem(inputtoprogram0,programid);
 980   recorded_measured_execution_times0[0] =
run_program_with_input_dont_use_filesystem(inputtoprogram0,programid);
 981   recorded_measured_execution_times1[0] =
run_program_with_input_dont_use_filesystem(inputtoprogram1,programid);
 982   recorded_measured_execution_times1[0] =
run_program_with_input_dont_use_filesystem(inputtoprogram1,programid);
 983   for (iterator=0;iterator<ntimes;iterator++) {
 984     recorded_measured_execution_times0[iterator] =
run_program_with_input_dont_use_filesystem(inputtoprogram0,programid);
 985     recorded_measured_execution_times0[iterator] =
run_program_with_input_dont_use_filesystem(inputtoprogram0,programid);
```

```
 986     recorded_measured_execution_times1[iterator] =
run_program_with_input_dont_use_filesystem(inputtoprogram1,programid);
 987     recorded_measured_execution_times1[iterator] =
run_program_with_input_dont_use_filesystem(inputtoprogram1,programid);
 988   }
 989
sort_recorded_measured_execution_times_ascending_order(recorded_measured_exec
ution_times0,ntimes);
 990
sort_recorded_measured_execution_times_ascending_order(recorded_measured_exec
ution_times1,ntimes);
 991   *p_t0 = recorded_measured_execution_times0[ntimes/2];
 992   *p_t1 = recorded_measured_execution_times1[ntimes/2];
 993   free(recorded_measured_execution_times0);
 994   free(recorded_measured_execution_times1);
 995 }
 996
 997 // *** END: Above shows code that is used for running target programs
 998
 999 struct onerun {
1000   int* inputtoprogram;
1001   double t;
1002   double error;
1003   double
recordedexecutiontimes[REPLICATION_TO_GET_MEDIAN_OF_EXECUTIONTIME_MEASUREMENT
S];
1004   double prediction; // this is a prediction of the execution time based
on weights and inputtoprogram
1005   double weight;     // we use this as weighting there are very little
data at extreme execution times; these should we weighted higher
1006 };
1007
1008 struct onerun* manyruns;
1009
1010 int* index_of_examples;
1011 int* selected_index_of_examples;
1012 int n_selected_index_of_examples;
1013 int* abs_error_index_of_examples;
1014
1015 double min_observed_time;
1016 double max_observed_time;
1017 double average_observed_time;
1018
1019 int predictedworstcaseinput[inputsizeinnumberofbits/((sizeof(int))*8)];
1020 double predicted_time_predictedworstcaseinput;
1021 double actual_time_predictedworstcaseinput;
1022
1023 double proposed_initial_w0[inputsizeinnumberofbits];
1024
1025 int* predictedworstcaseinput_phase0;
1026 int* predictedworstcaseinput_phase1;
1027
1028 double predicted_time_predictedworstcaseinput_phase0;
1029 double predicted_time_predictedworstcaseinput_phase1;
1030
1031 double* weights_for_regression_with_affine_function;
1032 double regression_with_affine_function_min_prediction;
```

```
1033 double regression_with_affine_function_max_prediction;
1034 double regression_with_affine_function_max_abs_error;
1035 int regression_with_affine_function_index_with_max_abs_error;
1036 double
regression_with_affine_function_prediction_of_example_with_max_abs_error;
1037 double regression_with_affine_function_t_of_example_with_max_abs_error;
1038 double regression_with_affine_function_average_abs_error;
1039 double regression_with_affine_function_average_abs_error_uniform_time;
// this is computed based on sampling
1040 double neural_network_average_abs_error_uniform_time; // this is
computed based on sampling
1041
1042 void
allocate_memory_for_predictedworstcaseinput_phase0_and_predictedworstcaseinpu
t_phase1() {
1043   int n_elements;
1044   n_elements = inputsizeinnumberofbits/((sizeof(int))*8);
1045   predictedworstcaseinput_phase0 = (int*) malloc( sizeof(int) *
n_elements );
1046   predictedworstcaseinput_phase1 = (int*) malloc( sizeof(int) *
n_elements );
1047 }
1048 void
free_memory_for_predictedworstcaseinput_phase0_and_predictedworstcaseinput_ph
ase1() {
1049   free(predictedworstcaseinput_phase0);
1050   free(predictedworstcaseinput_phase1);
1051 }
1052
1053 void allocate_memory_for_regression_with_affine_function() {
1054   weights_for_regression_with_affine_function   =
malloc(inputsizeinnumberofbits*(sizeof(double))); if
(weights_for_regression_with_affine_function==NULL) { printf("Memory
allocation failure in
allocate_memory_for_regression_with_affine_function.\n");   exit(-1); }
1055 }
1056 void free_memory_for_regression_with_affine_function() {
1057   free(weights_for_regression_with_affine_function);
1058 }
1059
1060 void allocate_memory_for_manyruns_v2(int n_examples) {
1061   int i;
1062   manyruns = (struct onerun*) malloc( sizeof(struct onerun) *
n_examples);
1063   if (manyruns==NULL) { printf("Error in
allocate_memory_for_manyruns_v2. Failed when allocating manyruns.\n"); exit(-
1); }
1064   for (i=0;i<n_examples;i++) {
1065     manyruns[i].inputtoprogram = (int*)
malloc(getinputtoprogram_size_in_number_of_bytes());
1066     if (manyruns[i].inputtoprogram==NULL) { printf("Error in
allocate_memory_for_manyruns_v2. Failed when allocating
manyruns[i].inputtoprogram.\n"); exit(-1); }
1067   }
1068 }
1069 void free_memory_for_manyruns_v2(int n_examples) {
1070   int i;
```

```
1071    for (i=0;i<n_examples;i++) {
1072      free(manyruns[i].inputtoprogram);
1073    }
1074    free(manyruns);
1075  }
1076
1077  void read_captured_data_from_disk_all_inputs_one_explicit_fn(char*
fn,int n_examples) {
1078    FILE* f; int i;
1079    f = fopen(fn, "rb");
1080    // if (f!=NULL) { printf("Opening file. success\n"); } else {
printf("Opening file. failed\n"); }
1081    if (f==NULL) { printf("In
read_captured_data_from_disk_all_inputs_one_explicit_fn. Opening file.
failed\n"); exit(-1); }
1082    for (i=0;i<n_examples;i++) {
1083      if (i%1000000==0) { printf("Reading file. i = %d\n", i); }
1084
fread(manyruns[i].inputtoprogram,sizeof(int),inputsizeinnumberofbits/(sizeof(
int)*8),f);
1085    }
1086    printf("Finished reading file.\n");
1087    fclose( f);
1088  }
1089
1090  void read_captured_data_from_disk_all_ts_one_explicit_fn(char* fn,int
n_examples) {
1091    FILE* f; int i;
1092    f = fopen(fn, "rb");
1093    // if (f!=NULL) { printf("Opening file. success\n"); } else {
printf("Opening file. failed\n"); }
1094    if (f==NULL) { printf("In
read_captured_data_from_disk_all_ts_one_explicit_fn. Opening file.
failed\n"); exit(-1); }
1095    for (i=0;i<n_examples;i++) {
1096      if (i%1000000==0) { printf("Reading file. i = %d\n", i); }
1097      fread(&(manyruns[i].t),sizeof(double),1,f);
1098    }
1099    printf("Finished reading file.\n");
1100    fclose( f);
1101  }
1102
1103  void read_captured_data_from_disk_two_explicit_fn(char*
fn_with_executiontimemeasurements_inputs,char*
fn_with_executiontimemeasurements_times,int n_examples) {
1104
read_captured_data_from_disk_all_inputs_one_explicit_fn(fn_with_executiontime
measurements_inputs,n_examples);
1105    read_captured_data_from_disk_all_ts_one_explicit_fn(
fn_with_executiontimemeasurements_times,n_examples);
1106  }
1107
1108  double calculate_median_from_double_array(double* p_doublearray,int
nelements) {
1109    int temp1; int temp2;
1110    if (nelements<=0) {
1111      printf("Error in calculate_median_from_double_array. No data.\n");
```

```
1112    } else {
1113      for (temp1=0;temp1<nelements;temp1++) {
1114        for (temp2=temp1+1;temp2<nelements;temp2++) {
1115          if (p_doublearray[temp1]>p_doublearray[temp2]) {
1116            swapdouble(&(p_doublearray[temp1]),&(p_doublearray[temp2]));
1117          }
1118        }
1119      }
1120    }
1121    return p_doublearray[nelements/2];
1122  }
1123
1124  int** experimentorder;
1125
1126  void shuffle_int_array(int* p_int,int n_examples) {
1127    int iterator1; int iterator2; int index1; int index2; int temp;
1128    for (iterator1=0;iterator1<n_examples;iterator1++) {
1129      for (iterator2=0;iterator2<10;iterator2++) {
1130        index1 = random()%n_examples;
1131        index2 = random()%n_examples;
1132        temp = p_int[index1];
1133        p_int[index1] = p_int[index2];
1134        p_int[index2] = temp;
1135      }
1136    }
1137  }
1138
1139  void do_data_capture_internal(struct onerun* p_runs,int programid,int
n_examples) {
1140    // int i; int j; double t;
1141    int i; int j; double t; int iterator_index;
1142    int replication_index;
1143    for (i=0;i<n_examples;i++) {
1144      fill_int_array_with_random_bits(currentinputtoprogram);
1145      copy_inputtoprogram(p_runs[i].inputtoprogram,
currentinputtoprogram);
1146    }
1147    experimentorder =
malloc(sizeof(int*)*REPLICATION_TO_GET_MEDIAN_OF_EXECUTIONTIME_MEASUREMENTS);
1148    for
(replication_index=0;replication_index<REPLICATION_TO_GET_MEDIAN_OF_EXECUTION
TIME_MEASUREMENTS;replication_index++) {
1149      experimentorder[replication_index] = malloc(sizeof(int)*n_examples);
1150    }
1151    for
(replication_index=0;replication_index<REPLICATION_TO_GET_MEDIAN_OF_EXECUTION
TIME_MEASUREMENTS;replication_index++) {
1152      for (i=0;i<n_examples;i++) {
1153        experimentorder[replication_index][i] = i;
1154      }
1155    }
1156    for
(replication_index=0;replication_index<REPLICATION_TO_GET_MEDIAN_OF_EXECUTION
TIME_MEASUREMENTS;replication_index++) {
1157      shuffle_int_array(experimentorder[replication_index],n_examples);
1158    }
```

```
1159   for
(replication_index=0;replication_index<REPLICATION_TO_GET_MEDIAN_OF_EXECUTION
TIME_MEASUREMENTS;replication_index++) {
1160     for (j=0;j<20;j++) {
1161       fill_int_array_with_random_bits(currentinputtoprogram);
1162       t =
run_program_with_input_dont_use_filesystem(currentinputtoprogram,programid);
// burn in
1163       t =
run_program_with_input_dont_use_filesystem(currentinputtoprogram,programid);
// burn in
1164     }
1165     for (iterator_index=0;iterator_index<n_examples;iterator_index++) {
1166       i = experimentorder[replication_index][iterator_index];
1167       copy_inputtoprogram(currentinputtoprogram,
p_runs[i].inputtoprogram);
1168       t =
run_program_with_input_dont_use_filesystem(currentinputtoprogram,programid);
// we run it twice but use only the result of the 2nd run; this helps to
eliminate potential effects of different initial states
1169       t =
run_program_with_input_dont_use_filesystem(currentinputtoprogram,programid);
1170       p_runs[i].recordedexecutiontimes[replication_index] = t;
1171     }
1172   }
1173   for (i=0;i<n_examples;i++) {
1174     p_runs[i].t = calculate_median_from_double_array(
p_runs[i].recordedexecutiontimes,
REPLICATION_TO_GET_MEDIAN_OF_EXECUTIONTIME_MEASUREMENTS );
1175   }
1176   for
(replication_index=0;replication_index<REPLICATION_TO_GET_MEDIAN_OF_EXECUTION
TIME_MEASUREMENTS;replication_index++) {
1177     free(experimentorder[replication_index]);
1178   }
1179   free(experimentorder);
1180 }
1181
1182 void do_data_capture(int programid,int n_examples) {
1183   do_data_capture_internal(manyruns,programid,n_examples);
1184 }
1185
1186 void write_captured_data_from_disk_all_inputs_one_explicit_fn(char*
fn,int n_examples) {
1187   FILE* f; int i;
1188   f = fopen(fn, "wb");
1189   for (i=0;i<n_examples;i++) {
1190
fwrite(manyruns[i].inputtoprogram,sizeof(int),inputsizeinnumberofbits/(sizeof
(int)*8),f);
1191   }
1192   fclose( f);
1193 }
1194
1195 void write_captured_data_from_disk_all_ts_one_explicit_fn(char* fn,int
n_examples) {
1196   FILE* f; int i;
```

```
1197   f = fopen(fn, "wb");
1198   for (i=0;i<n_examples;i++) {
1199     fwrite(&(manyruns[i].t),sizeof(double),1,f);
1200   }
1201   fclose( f);
1202 }
1203
1204 void write_captured_data_to_disk_two_explicit_fn(char*
fn_with_executiontimemeasurements_inputs,char*
fn_with_executiontimemeasurements_times,int n_examples) {
1205
write_captured_data_from_disk_all_inputs_one_explicit_fn(fn_with_executiontim
emeasurements_inputs,n_examples);
1206   write_captured_data_from_disk_all_ts_one_explicit_fn(
fn_with_executiontimemeasurements_times,n_examples);
1207 }
1208
1209 #define HIDDEN_DIM 64
1210
1211 typedef struct model_t {
1212   TF_Graph* graph;
1213   TF_Session* session;
1214   TF_Status* status;
1215
1216   TF_Output input, target, output;
1217
1218   TF_Operation *init_op, *train_op, *save_op, *restore_op;
1219   TF_Output checkpoint_file;
1220 } model_t;
1221
1222 double* p_w0_value;
1223 double* p_w1_value;
1224 double* p_w2_value;
1225 double* p_w3_value;
1226 double* p_w4_value;
1227 double* p_b1_value;
1228 double* p_b2_value;
1229 double* p_b3_value;
1230 double* p_b4_value;
1231 double* p_w5_value;
1232 double* p_w6_value;
1233 double* p_b56_value;
1234 double* p_loss_value;
1235
1236 void allocate_memory_for_learnable_weights() {
1237   p_w0_value  = malloc((sizeof(double))*inputsizeinnumberofbits);
1238   p_w1_value  =
malloc((sizeof(double))*inputsizeinnumberofbits*HIDDEN_DIM);
1239   p_w2_value  =
malloc((sizeof(double))*inputsizeinnumberofbits*HIDDEN_DIM);
1240   p_w3_value  =
malloc((sizeof(double))*inputsizeinnumberofbits*HIDDEN_DIM);
1241   p_w4_value  =
malloc((sizeof(double))*inputsizeinnumberofbits*HIDDEN_DIM);
1242   p_b1_value  = malloc((sizeof(double))*HIDDEN_DIM);
1243   p_b2_value  = malloc((sizeof(double))*HIDDEN_DIM);
1244   p_b3_value  = malloc((sizeof(double))*HIDDEN_DIM);
```

```
1245   p_b4_value  = malloc((sizeof(double))*HIDDEN_DIM);
1246   p_w5_value  = malloc((sizeof(double))*HIDDEN_DIM);
1247   p_w6_value  = malloc((sizeof(double))*HIDDEN_DIM);
1248   p_b56_value = malloc(sizeof(double));
1249   p_loss_value = malloc(sizeof(double));
1250 }
1251
1252 void free_memory_for_learnable_weights() {
1253   free( p_w0_value);
1254   free( p_w1_value);
1255   free( p_w2_value);
1256   free( p_w3_value);
1257   free( p_w4_value);
1258   free( p_b1_value);
1259   free( p_b2_value);
1260   free( p_b3_value);
1261   free( p_b4_value);
1262   free( p_w5_value);
1263   free( p_w6_value);
1264   free( p_b56_value);
1265   free( p_loss_value);
1266 }
1267
1268 void allocate_memory_for_learnable_weights_v2() {
1269   p_w0_value  = malloc((sizeof(double))*inputsizeinnumberofbits);
1270   p_b56_value = malloc(sizeof(double));
1271 }
1272
1273 void free_memory_for_learnable_weights_v2() {
1274   free( p_w0_value);
1275   free( p_b56_value);
1276 }
1277
1278 int Okay(TF_Status* status) {
1279   if (TF_GetCode(status) != TF_OK) {
1280     fprintf(stderr, "ERROR: %s\n", TF_Message(status));
1281     return 0;
1282   }
1283   return 1;
1284 }
1285
1286 int get_weight_general(model_t* p_model,double* p,char*
readopstring,int64_t* dims_out,int ndim,size_t nbytes_out) {
1287   TF_Tensor* t_out = TF_AllocateTensor(TF_DOUBLE, dims_out, ndim,
nbytes_out);
1288   TF_Output outputs[1];
1289   outputs[0].oper = TF_GraphOperationByName(p_model-
>graph,readopstring);
1290   outputs[0].index = 0;
1291   TF_Tensor* output_values[1];
1292   output_values[0] = t_out;
1293   TF_SessionRun(p_model->session, NULL, NULL, NULL, 0, outputs,
1294                 output_values, 1,
1295                 NULL, 0, NULL, p_model->status);
1296   if (!Okay(p_model->status)) return 0;
1297   if (TF_TensorByteSize(output_values[0]) != nbytes_out) {
1298     fprintf(stderr,
```

```
1299              "ERROR: Expected predictions tensor to have %zu bytes, has
%zu\n",
1300              nbytes_out, TF_TensorByteSize(output_values[0]));
1301     TF_DeleteTensor(output_values[0]);
1302     return 0;
1303   }
1304   memcpy(p, TF_TensorData(output_values[0]), nbytes_out);
1305   TF_DeleteTensor(output_values[0]);
1306   return 1;
1307 }
1308
1309 int get_weight_single_element(model_t* p_model,double* p,char*
readopstring) {
1310   return
get_weight_general(p_model,p,readopstring,NULL,0,sizeof(double));
1311 }
1312
1313 int get_weight_vector(model_t* p_model,double* p,char* readopstring,int
n_elements) {
1314   int64_t dims_out[1] = {n_elements};
1315   return
get_weight_general(p_model,p,readopstring,dims_out,1,(sizeof(double))*n_eleme
nts);
1316 }
1317
1318 int get_weight_matrix(model_t* p_model,double* p,char* readopstring,int
n_rows,int n_columns) {
1319   int64_t dims_out[2] = {n_rows,n_columns};
1320   return
get_weight_general(p_model,p,readopstring,dims_out,2,(sizeof(double))*n_rows*
n_columns);
1321 }
1322
1323 int get_weight_w0(model_t* p_model,double* p)  { return
get_weight_vector(p_model,p,"w0/Read/ReadVariableOp",inputsizeinnumberofbits)
; }
1324 int get_weight_w1(model_t* p_model,double* p)  { return
get_weight_matrix(p_model,p,"w1/Read/ReadVariableOp",inputsizeinnumberofbits,
HIDDEN_DIM); }
1325 int get_weight_w2(model_t* p_model,double* p)  { return
get_weight_matrix(p_model,p,"w2/Read/ReadVariableOp",inputsizeinnumberofbits,
HIDDEN_DIM); }
1326 int get_weight_w3(model_t* p_model,double* p)  { return
get_weight_matrix(p_model,p,"w3/Read/ReadVariableOp",inputsizeinnumberofbits,
HIDDEN_DIM); }
1327 int get_weight_w4(model_t* p_model,double* p)  { return
get_weight_matrix(p_model,p,"w4/Read/ReadVariableOp",inputsizeinnumberofbits,
HIDDEN_DIM); }
1328 int get_weight_b1(model_t* p_model,double* p)  { return
get_weight_vector(p_model,p,"b1/Read/ReadVariableOp",HIDDEN_DIM);       }
1329 int get_weight_b2(model_t* p_model,double* p)  { return
get_weight_vector(p_model,p,"b2/Read/ReadVariableOp",HIDDEN_DIM);       }
1330 int get_weight_b3(model_t* p_model,double* p)  { return
get_weight_vector(p_model,p,"b3/Read/ReadVariableOp",HIDDEN_DIM);       }
1331 int get_weight_b4(model_t* p_model,double* p)  { return
get_weight_vector(p_model,p,"b4/Read/ReadVariableOp",HIDDEN_DIM);       }
```

```
1332 int get_weight_w5(model_t* p_model,double* p)  { return
get_weight_vector(p_model,p,"w5/Read/ReadVariableOp",HIDDEN_DIM);       }
1333 int get_weight_w6(model_t* p_model,double* p)  { return
get_weight_vector(p_model,p,"w6/Read/ReadVariableOp",HIDDEN_DIM);       }
1334 int get_weight_b56(model_t* p_model,double* p) { return
get_weight_single_element(p_model,p,"b56/Read/ReadVariableOp");         }
1335 int get_loss(model_t* p_model,double* p) { return
get_weight_single_element(p_model,p,"loss"); }
1336
1337 void fill_memory_with_learnable_weights(model_t* p_model) {
1338   if (!get_weight_w0(p_model,p_w0_value)) exit(-1);
1339   if (!get_weight_w1(p_model,p_w1_value)) exit(-1);
1340   if (!get_weight_w2(p_model,p_w2_value)) exit(-1);
1341   if (!get_weight_w3(p_model,p_w3_value)) exit(-1);
1342   if (!get_weight_w4(p_model,p_w4_value)) exit(-1);
1343   if (!get_weight_b1(p_model,p_b1_value)) exit(-1);
1344   if (!get_weight_b2(p_model,p_b2_value)) exit(-1);
1345   if (!get_weight_b3(p_model,p_b3_value)) exit(-1);
1346   if (!get_weight_b4(p_model,p_b4_value)) exit(-1);
1347   if (!get_weight_w5(p_model,p_w5_value)) exit(-1);
1348   if (!get_weight_w6(p_model,p_w6_value)) exit(-1);
1349   if (!get_weight_b56(p_model,p_b56_value)) exit(-1);
1350   // if (!get_loss(p_model,p_loss_value)) exit(-1); // this function
does not work; so we don't call it
1351 }
1352
1353 int get_input_dim() {
1354   return inputsizeinnumberofbits;
1355 }
1356
1357 void fill_in_test_data_make_all_random(double* testdata,int
n_testcases,int n_elements_in_input) {
1358   int i; int j;
1359   for (i=0;i<n_testcases;i++) {
1360     for (j=0;j<n_elements_in_input;j++) {
1361       if ((random()%2)==1) {
1362         testdata[i*n_elements_in_input+j] = 1.0;
1363       } else {
1364         testdata[i*n_elements_in_input+j] = 0.0;
1365       }
1366     }
1367   }
1368 }
1369 void fill_in_test_data_set_all_elements_in_row_equal_to(double*
testdata,int n_testcases,int n_elements_in_input,int rowindex,double
set_to_value) {
1370   int j;
1371   for (j=0;j<n_elements_in_input;j++) {
1372     testdata[rowindex*n_elements_in_input+j] = set_to_value;
1373   }
1374 }
1375
1376 void fill_in_test_data_set_ascending(double* testdata,int
n_testcases,int n_elements_in_input,int largestep,int rowindex) {
1377   int j; int considered_integer; int bits_per_int; int step;
1378   bits_per_int = 8*(sizeof(int));
1379   if (largestep) {
```

```
1380      considered_integer = -2147483648;
1381      step = 4294967296.0 / (n_elements_in_input/bits_per_int); // we
represent 4294967296 as floating point in order to make sure we don't end up
with overflow
1382    } else {
1383      considered_integer = -(n_elements_in_input/bits_per_int)/2;
1384      step = 1;
1385    }
1386    for (j=0;j<n_elements_in_input;j++) {
1387      if ((considered_integer >> (j%bits_per_int)) & 1) {
1388        testdata[rowindex*n_elements_in_input+j] = 1.0;
1389      } else {
1390        testdata[rowindex*n_elements_in_input+j] = 0.0;
1391      }
1392      if ((j%bits_per_int)==bits_per_int-1) {
1393        considered_integer = considered_integer + step;
1394      }
1395    }
1396 }
1397
1398 void fill_in_test_data_set_descending(double* testdata,int
n_testcases,int n_elements_in_input,int largestep,int rowindex) {
1399    int j; int considered_integer; int bits_per_int; int step;
1400    bits_per_int = 8*(sizeof(int));
1401    if (largestep) {
1402      considered_integer = 2147483647;
1403      step = -4294967296.0 / (n_elements_in_input/bits_per_int); // we
represent -4294967296 as floating point in order to make sure we don't end up
with overflow
1404    } else {
1405      considered_integer = (n_elements_in_input/bits_per_int)/2;
1406      step = -1;
1407    }
1408    for (j=0;j<n_elements_in_input;j++) {
1409      if ((considered_integer >> (j%bits_per_int)) & 1) {
1410        testdata[rowindex*n_elements_in_input+j] = 1.0;
1411      } else {
1412        testdata[rowindex*n_elements_in_input+j] = 0.0;
1413      }
1414      if ((j%bits_per_int)==bits_per_int-1) {
1415        considered_integer = considered_integer + step;
1416      }
1417    }
1418 }
1419
1420 void fill_in_test_data_basic(double* testdata,int n_testcases,int
n_elements_in_input) {
1421
fill_in_test_data_make_all_random(testdata,n_testcases,n_elements_in_input);
1422
fill_in_test_data_set_all_elements_in_row_equal_to(testdata,n_testcases,n_ele
ments_in_input,0,0.0);
1423
fill_in_test_data_set_all_elements_in_row_equal_to(testdata,n_testcases,n_ele
ments_in_input,1,1.0);
1424 }
```

```
1425 void fill_in_test_data_used_for_bubblesort(double* testdata,int
n_testcases,int n_elements_in_input) {
1426   int j;
1427
fill_in_test_data_make_all_random(testdata,n_testcases,n_elements_in_input);
1428   fill_in_test_data_set_ascending(
testdata,n_testcases,n_elements_in_input,0,0);
1429
fill_in_test_data_set_descending(testdata,n_testcases,n_elements_in_input,0,1
);
1430   fill_in_test_data_set_ascending(
testdata,n_testcases,n_elements_in_input,1,2);
1431
fill_in_test_data_set_descending(testdata,n_testcases,n_elements_in_input,1,3
);
1432 }
1433 void fill_in_test_data(double* testdata,int n_testcases,int
n_elements_in_input) {
1434   // fill_in_test_data_basic(testdata,n_testcases,n_elements_in_input);
1435
fill_in_test_data_used_for_bubblesort(testdata,n_testcases,n_elements_in_inpu
t);
1436 }
1437
1438 void calculate_min_max_average_executiontime_from_manyruns(int
n_examples) {
1439   int i;
1440   if (n_examples<=0) { printf("Error in
calculate_min_max_average_executiontime_from_manyruns\n"); exit(-1); }
1441   min_observed_time     = manyruns[0].t;
1442   max_observed_time     = manyruns[0].t;
1443   average_observed_time = manyruns[0].t;
1444   for (i=1;i<n_examples;i++) {
1445     if (min_observed_time>manyruns[i].t) { min_observed_time =
manyruns[i].t; }
1446     if (max_observed_time<manyruns[i].t) { max_observed_time =
manyruns[i].t; }
1447     average_observed_time = average_observed_time + manyruns[i].t;
1448   }
1449   average_observed_time = average_observed_time / n_examples;
1450   printf("min_observed_time = %lf\n",     min_observed_time);
1451   printf("max_observed_time = %lf\n",     max_observed_time);
1452   printf("average_observed_time = %lf\n", average_observed_time);
1453 }
1454
1455 void get_min_max_average_execution_time(double* p_mint,double*
p_maxt,double* p_averaget) {
1456   *p_mint     = min_observed_time;
1457   *p_maxt     = max_observed_time;
1458   *p_averaget = average_observed_time;
1459 }
1460
1461 int
obtain_row_using_uniform_sampling_over_execution_time_given_lb_index_and_ub_i
ndex_of_sorted_order(int lb_index,int ub_index) {
1462   double target_t; double t_lb_index; double t_ub_index; int
index_middle; double t_middle;
```

```
1463    t_lb_index = manyruns[index_of_examples[lb_index]].t;
1464    t_ub_index = manyruns[index_of_examples[ub_index]].t;
1465    target_t = t_lb_index + (t_ub_index-t_lb_index)*drand48();
1466    printf("In
obtain_row_using_uniform_sampling_over_execution_time_given_lb_index_and_ub_i
ndex_of_sorted_order. lb_index=%d ub_index=%d t_lb_index=%lf t_ub_index=%lf
target_t = %lf\n", lb_index, ub_index, t_lb_index, t_ub_index, target_t);
1467    while (lb_index<ub_index) {
1468      printf("In
obtain_row_using_uniform_sampling_over_execution_time_given_lb_index_and_ub_i
ndex_of_sorted_order. lb_index=%d ub_index=%d\n", lb_index, ub_index);
1469      if (lb_index+1==ub_index) {
1470        t_middle = (t_lb_index+t_ub_index)/2;
1471        if (target_t<t_middle) {
1472          ub_index = ub_index - 1;
1473        } else {
1474          lb_index = lb_index + 1;
1475        }
1476      } else {
1477        index_middle = (lb_index+ub_index)/2;
1478        t_middle = manyruns[index_of_examples[index_middle]].t;
1479        if (target_t<t_middle) {
1480          ub_index = index_middle;
1481          t_ub_index = t_middle;
1482        } else {
1483          lb_index = index_middle;
1484          t_lb_index = t_middle;
1485        }
1486      }
1487    }
1488    return index_of_examples[lb_index];
1489  }
1490
1491  int obtain_row_using_uniform_sampling_over_execution_time(int
n_examples) {
1492    return
obtain_row_using_uniform_sampling_over_execution_time_given_lb_index_and_ub_i
ndex_of_sorted_order(0,n_examples-1);
1493  }
1494
1495  int
obtain_row_using_uniform_sampling_given_lb_index_and_ub_index_of_sorted_order
(int lb_index,int ub_index) {
1496    int tempindex;
1497    tempindex = lb_index + (rand()%(ub_index-lb_index));
1498    return index_of_examples[tempindex];
1499  }
1500
1501  void print_min_max_average_execution_time() {
1502    double mint; double maxt; double averaget;
1503    get_min_max_average_execution_time(&mint,&maxt,&averaget);
1504    printf("mint = %lf maxt = %lf averaget = %lf\n",mint,maxt,averaget);
1505  }
1506
1507  int* array_of_row_indices;
1508  void allocate_memory_array_of_indices() { array_of_row_indices =
malloc(sizeof(int)*NUMBER_OF_CONSTRAINTS_FOR_MILP_INITIALIZING_W5_AND_W6); if
```

```
(array_of_row_indices==NULL) { printf("Error in
allocate_memory_array_of_indices\n"); exit(-1); } }
1509 void free_memory_array_of_row_indices() { free(array_of_row_indices); }
1510
1511 // if n_examples==NUMBER_OF_CONSTRAINTS_FOR_MILP_INITIALIZING_W5_AND_W6,
then we may want to rewrite this so that it gets simpler
1512 void fill_array_of_row_indices(int n_examples) {
1513   int lo_index; int hi_index; int n_elements_added; int row_in_dataset;
1514   lo_index = 0;
1515   hi_index = n_examples-1;
1516   n_elements_added = 0;
1517   while
(n_elements_added<NUMBER_OF_CONSTRAINTS_FOR_MILP_INITIALIZING_W5_AND_W6/2) {
1518     if (lo_index<=hi_index) {
1519       if ((n_elements_added%2)==0) {
1520         array_of_row_indices[n_elements_added] =
index_of_examples[lo_index];
1521         lo_index = lo_index + 1;
1522       } else {
1523         array_of_row_indices[n_elements_added] =
index_of_examples[hi_index];
1524         hi_index = hi_index - 1;
1525       }
1526       n_elements_added = n_elements_added + 1;
1527     } else {
1528       printf("Error in fill_array_of_row_indices\n"); exit(-1);
1529     }
1530   }
1531   while
(n_elements_added<NUMBER_OF_CONSTRAINTS_FOR_MILP_INITIALIZING_W5_AND_W6) {
1532     row_in_dataset =
obtain_row_using_uniform_sampling_over_execution_time(n_examples);
1533     array_of_row_indices[n_elements_added] = row_in_dataset;
1534     n_elements_added = n_elements_added + 1;
1535   }
1536 }
1537
1538 struct data_for_w5_and_w6_leq_constraint {
1539   double w5[HIDDEN_DIM];
1540   double w6[HIDDEN_DIM];
1541   double rhs;
1542 };
1543
1544 struct data_for_w5_and_w6_leq_constraint*
array_with_data_for_w5_and_w6_leq_constraints;
1545 void allocate_memory_array_with_data_for_w5_and_w6_leq_constraints() {
1546   array_with_data_for_w5_and_w6_leq_constraints = malloc(sizeof(struct
data_for_w5_and_w6_leq_constraint)*NUMBER_OF_CONSTRAINTS_FOR_MILP_INITIALIZIN
G_W5_AND_W6);  if (array_with_data_for_w5_and_w6_leq_constraints==NULL) {
printf("Error in
allocate_memory_array_with_data_for_w5_and_w6_leq_constraints\n"); exit(-1);
}
1547 }
1548 void free_memory_array_with_data_for_w5_and_w6_constraints() {
1549   free(array_with_data_for_w5_and_w6_leq_constraints);
1550 }
```

```
1551 void setup_constraint_error(int index_of_w5_w6_constraints,int
row_in_dataset) {
1552   int j; int k;
1553   double sum1[HIDDEN_DIM]; double sum2[HIDDEN_DIM]; double
sum3[HIDDEN_DIM]; double sum4[HIDDEN_DIM];
1554   double mint; double maxt; double averaget;
1555   double sum;
1556   for (k=0;k<HIDDEN_DIM;k++) { sum1[k] = p_b1_value[k]; sum2[k] =
p_b2_value[k]; sum3[k] = p_b3_value[k]; sum4[k] = p_b4_value[k]; }
1557   get_min_max_average_execution_time(&mint,&maxt,&averaget);
1558   sum = averaget;
1559   for (j=0;j<inputsizeinnumberofbits;j++) {
1560     if
(isbitonein_bitposition_in_integerarray(manyruns[row_in_dataset].inputtoprogr
am,j)) {
1561       for (k=0;k<HIDDEN_DIM;k++) {
1562         sum1[k] = sum1[k] + p_w1_value[j*HIDDEN_DIM+k];
1563         sum2[k] = sum2[k] + p_w2_value[j*HIDDEN_DIM+k];
1564         sum3[k] = sum3[k] + p_w3_value[j*HIDDEN_DIM+k];
1565         sum4[k] = sum4[k] + p_w4_value[j*HIDDEN_DIM+k];
1566       }
1567       sum = sum + weights_for_regression_with_affine_function[j];
1568     } else {
1569       for (k=0;k<HIDDEN_DIM;k++) {
1570         sum1[k] = sum1[k] - p_w1_value[j*HIDDEN_DIM+k];
1571         sum2[k] = sum2[k] - p_w2_value[j*HIDDEN_DIM+k];
1572         sum3[k] = sum3[k] - p_w3_value[j*HIDDEN_DIM+k];
1573         sum4[k] = sum4[k] - p_w4_value[j*HIDDEN_DIM+k];
1574       }
1575       sum = sum - weights_for_regression_with_affine_function[j];
1576     }
1577   }
1578   for (k=0;k<HIDDEN_DIM;k++) {
1579
array_with_data_for_w5_and_w6_leq_constraints[index_of_w5_w6_constraints].w5[
k] = mindouble(sum1[k],sum2[k]);
1580
array_with_data_for_w5_and_w6_leq_constraints[index_of_w5_w6_constraints].w6[
k] = maxdouble(sum3[k],sum4[k]);
1581   }
1582   sum = sum - manyruns[row_in_dataset].t;
1583
array_with_data_for_w5_and_w6_leq_constraints[index_of_w5_w6_constraints].rhs
= -sum;
1584 }
1585
1586 struct myargstruct2 {
1587   int loindex;
1588   int hiindex;
1589 };
1590 pthread_t threads2[NUM_THREADS_USED_FOR_WORK];
1591 pthread_attr_t attrs2[NUM_THREADS_USED_FOR_WORK];
1592 struct myargstruct2 myargstruct_array2[NUM_THREADS_USED_FOR_WORK];
1593
1594 void set_indices_in_myargstruct_array2(int th) {
1595   int objects_per_worker;
```

```
1596   if
(NUMBER_OF_CONSTRAINTS_FOR_MILP_INITIALIZING_W5_AND_W6%NUM_THREADS_USED_FOR_W
ORK==0) {
1597     objects_per_worker =
NUMBER_OF_CONSTRAINTS_FOR_MILP_INITIALIZING_W5_AND_W6/NUM_THREADS_USED_FOR_WO
RK;
1598   } else {
1599     objects_per_worker =
NUMBER_OF_CONSTRAINTS_FOR_MILP_INITIALIZING_W5_AND_W6/NUM_THREADS_USED_FOR_WO
RK + 1;
1600   }
1601   myargstruct_array2[th].loindex = th    * objects_per_worker;
1602   myargstruct_array2[th].hiindex = (th+1)* objects_per_worker - 1;
1603   if
(myargstruct_array2[th].hiindex>NUMBER_OF_CONSTRAINTS_FOR_MILP_INITIALIZING_W
5_AND_W6-1) {
1604     myargstruct_array2[th].hiindex =
NUMBER_OF_CONSTRAINTS_FOR_MILP_INITIALIZING_W5_AND_W6-1;
1605   }
1606 }
1607
1608 __attribute__((optimize("-O3"))) void* worker2(void *p) {
1609   int i;
1610   for (i=((struct myargstruct2*) p)->loindex;i<=((struct myargstruct2*)
p)->hiindex;i++) {
1611     setup_constraint_error(i,array_of_row_indices[i]);
1612   }
1613   pthread_exit(NULL);
1614 }
1615
1616 void fill_array_with_data_for_w5_and_w6_leq_constraints() {
1617   int rc; int th; void* status;
1618   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
1619      pthread_attr_init(&(attrs2[th]));
1620      pthread_attr_setdetachstate(&(attrs2[th]),PTHREAD_CREATE_JOINABLE);
1621      set_indices_in_myargstruct_array2(th);
1622      rc = pthread_create(&(threads2[th]),&(attrs2[th]),worker2,(void*)
(&(myargstruct_array2[th])));
1623      if (rc){
1624         printf("ERROR; return code from pthread_create() is %d\n", rc);
1625         exit(-1);
1626      }
1627   }
1628   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
1629      rc = pthread_join(threads2[th], &status);
1630      if (rc) {
1631         printf("ERROR; return code from pthread_join() is %d\n", rc);
1632         exit(-1);
1633      }
1634   }
1635   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
1636     pthread_attr_destroy( &(attrs2[th]) );
1637   }
1638 }
1639
1640 void compute_count_and_sum_uniform(int n_examples,int j,int lo_index,int
hi_index,int* p_count0,double* p_sum0,int* p_count1,double* p_sum1) {
```

```
1641   int iterator; int row_in_dataset; int count0; double sum0; int count1;
double sum1;
1642   count0=0; sum0 = 0.0; count1=0; sum1 = 0.0;
1643   iterator=lo_index;
1644   while (iterator<=hi_index) {
1645     row_in_dataset = iterator;
1646     if
(isbitonein_bitposition_in_integerarray(manyruns[row_in_dataset].inputtoprogr
am,j)) {
1647       count1 = count1 + 1;
1648       sum1   = sum1   + manyruns[row_in_dataset].error;
1649     } else {
1650       count0 = count0 + 1;
1651       sum0   = sum0   + manyruns[row_in_dataset].error;
1652     }
1653     iterator++;
1654   }
1655   *p_count0 = count0;
1656   *p_sum0   = sum0;
1657   *p_count1 = count1;
1658   *p_sum1   = sum1;
1659 }
1660 void compute_count_and_sum_uniform_time(int n_examples,int j,int
lo_index,int hi_index,int* p_count0,double* p_sum0,int* p_count1,double*
p_sum1) {
1661   int iterator; int row_in_dataset; int count0; double sum0; int count1;
double sum1;
1662   count0=0; sum0 = 0.0; count1=0; sum1 = 0.0;
1663   iterator=lo_index;
1664   while (iterator<=hi_index) {
1665     row_in_dataset =
obtain_row_using_uniform_sampling_over_execution_time(n_examples);
1666     if
(isbitonein_bitposition_in_integerarray(manyruns[row_in_dataset].inputtoprogr
am,j)) {
1667       count1 = count1 + 1;
1668       sum1   = sum1   + manyruns[row_in_dataset].error;
1669     } else {
1670       count0 = count0 + 1;
1671       sum0   = sum0   + manyruns[row_in_dataset].error;
1672     }
1673     // iterator+=10; // ideally we should only increment by one but it
takes too much time; incrementing by 10 makes it run faster at the expense of
slightly worse estimation
1674     // iterator+=100; // ideally we should only increment by one but it
takes too much time; incrementing by 100 makes it run faster at the expense
of slightly worse estimation
1675     iterator+=10; // ideally we should only increment by one but it
takes too much time; incrementing by 10 makes it run faster at the expense of
slightly worse estimation
1676   }
1677   *p_count0 = count0;
1678   *p_sum0   = sum0;
1679   *p_count1 = count1;
1680   *p_sum1   = sum1;
1681 }
```

```
1682 void update_prediction_and_error_single_bit(int j,int lo_index,int
hi_index) {
1683   int row_in_dataset;
1684   for
(row_in_dataset=lo_index;row_in_dataset<=hi_index;row_in_dataset++) {
1685     if
(isbitonein_bitposition_in_integerarray(manyruns[row_in_dataset].inputtoprogr
am,j)) {
1686       manyruns[row_in_dataset].prediction =
manyruns[row_in_dataset].prediction +
weights_for_regression_with_affine_function[j];
1687     } else {
1688       manyruns[row_in_dataset].prediction =
manyruns[row_in_dataset].prediction -
weights_for_regression_with_affine_function[j];
1689     }
1690     manyruns[row_in_dataset].error = manyruns[row_in_dataset].prediction
- manyruns[row_in_dataset].t;
1691   }
1692 }
1693 void compute_prediction_and_error_all_bits(int lo_index,int hi_index) {
1694   double mint; double maxt; double averaget; int row_in_dataset; int j;
1695   get_min_max_average_execution_time(&mint,&maxt,&averaget);
1696   for
(row_in_dataset=lo_index;row_in_dataset<=hi_index;row_in_dataset++) {
1697     manyruns[row_in_dataset].prediction = averaget;
1698     for (j=0;j<inputsizeinnumberofbits;j++) {
1699       if
(isbitonein_bitposition_in_integerarray(manyruns[row_in_dataset].inputtoprogr
am,j)) {
1700         manyruns[row_in_dataset].prediction =
manyruns[row_in_dataset].prediction +
weights_for_regression_with_affine_function[j];
1701       } else {
1702         manyruns[row_in_dataset].prediction =
manyruns[row_in_dataset].prediction -
weights_for_regression_with_affine_function[j];
1703       }
1704     }
1705     manyruns[row_in_dataset].error = manyruns[row_in_dataset].prediction
- manyruns[row_in_dataset].t;
1706   }
1707 }
1708
1709 // the variable below can be understood as follows:
1710 //   input:
1711 //     int computation_to_perform;
1712 //     int n_examples;
1713 //     int j;
1714 //     int loindex;
1715 //     int hiindex;
1716 //   output
1717 //     int count0; double sum0; int count1; double sum1;
1718
1719 struct myargstruct {
1720   int computation_to_perform;
1721   int n_examples;
```

```
1722   int j;
1723   int loindex;
1724   int hiindex;
1725   int count0; double sum0; int count1; double sum1;
1726 };
1727 pthread_t threads[NUM_THREADS_USED_FOR_WORK];
1728 pthread_attr_t attrs[NUM_THREADS_USED_FOR_WORK];
1729 struct myargstruct myargstruct_array[NUM_THREADS_USED_FOR_WORK];
1730
1731 void set_indices_in_myargstruct_array(int th,int incoming_lo_index,int
incoming_hi_index,int computation_to_perform,int n_examples,int j) {
1732   int nobjects;
1733   int objects_per_worker;
1734   myargstruct_array[th].computation_to_perform = computation_to_perform;
1735   myargstruct_array[th].n_examples = n_examples;
1736   myargstruct_array[th].j = j;
1737   nobjects = (incoming_hi_index-incoming_lo_index+1);
1738   if (nobjects%NUM_THREADS_USED_FOR_WORK==0) {
1739     objects_per_worker = nobjects/NUM_THREADS_USED_FOR_WORK;
1740   } else {
1741     objects_per_worker = nobjects/NUM_THREADS_USED_FOR_WORK + 1;
1742   }
1743   myargstruct_array[th].loindex = incoming_lo_index+ th    *
objects_per_worker;
1744   myargstruct_array[th].hiindex = incoming_lo_index+ (th+1)*
objects_per_worker - 1;
1745   if (myargstruct_array[th].hiindex>incoming_hi_index) {
1746     myargstruct_array[th].hiindex = incoming_hi_index;
1747   }
1748 }
1749
1750 __attribute__((optimize("-O3"))) void* worker(void *p) {
1751   if (((struct myargstruct*) p)-
>computation_to_perform==COMPUTATION_TO_PERFORM_IS_COMPUTE_COUNT_AND_SUM_UNIF
ORM) {
1752     compute_count_and_sum_uniform(
1753       ((struct myargstruct*) p)->n_examples,
1754       ((struct myargstruct*) p)->j, ((struct myargstruct*) p)->loindex,
((struct myargstruct*) p)->hiindex,
1755       &(((struct myargstruct*) p)->count0), &(((struct myargstruct*) p)-
>sum0), &(((struct myargstruct*) p)->count1), &(((struct myargstruct*) p)-
>sum1) );
1756   } else {
1757     if (((struct myargstruct*) p)-
>computation_to_perform==COMPUTATION_TO_PERFORM_IS_COMPUTE_COUNT_AND_SUM_UNIF
ORM_TIME) {
1758       compute_count_and_sum_uniform_time(
1759         ((struct myargstruct*) p)->n_examples,
1760         ((struct myargstruct*) p)->j, ((struct myargstruct*) p)-
>loindex, ((struct myargstruct*) p)->hiindex,
1761         &(((struct myargstruct*) p)->count0), &(((struct myargstruct*)
p)->sum0), &(((struct myargstruct*) p)->count1), &(((struct myargstruct*) p)-
>sum1) );
1762     } else {
1763       if (((struct myargstruct*) p)-
>computation_to_perform==COMPUTATION_TO_PERFORM_IS_UPDATE_PREDICTION_AND_ERRO
R_SINGLE_BIT) {
```

```
1764          update_prediction_and_error_single_bit(
1765            ((struct myargstruct*) p)->j,((struct myargstruct*) p)-
>loindex,((struct myargstruct*) p)->hiindex);
1766        } else {
1767          if (((struct myargstruct*) p)-
>computation_to_perform==COMPUTATION_TO_PERFORM_IS_COMPUTE_PREDICTION_AND_ERR
OR_ALL_BITS) {
1768            compute_prediction_and_error_all_bits(((struct myargstruct*)
p)->loindex,((struct myargstruct*) p)->hiindex);
1769          } else {
1770            printf("Error in worker.\n"); exit(-1);
1771          }
1772        }
1773      }
1774    }
1775    pthread_exit(NULL);
1776 }
1777
1778 void compute_count_and_sum_uniform_do_parallel(int n_examples,int j,int
lo_index,int hi_index,int* p_count0,double* p_sum0,int* p_count1,double*
p_sum1) {
1779    int rc; int th; void* status;
1780    for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
1781       pthread_attr_init(&(attrs[th]));
1782       pthread_attr_setdetachstate(&(attrs[th]),PTHREAD_CREATE_JOINABLE);
1783
set_indices_in_myargstruct_array(th,lo_index,hi_index,COMPUTATION_TO_PERFORM_
IS_COMPUTE_COUNT_AND_SUM_UNIFORM,n_examples,j);
1784       rc = pthread_create(&(threads[th]),&(attrs[th]),worker,(void*)
(&(myargstruct_array[th])));
1785       if (rc){
1786          printf("ERROR; return code from pthread_create() is %d\n", rc);
1787          exit(-1);
1788       }
1789    }
1790    for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
1791       rc = pthread_join(threads[th], &status);
1792       if (rc) {
1793          printf("ERROR; return code from pthread_join() is %d\n", rc);
1794          exit(-1);
1795       }
1796    }
1797    for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
1798      pthread_attr_destroy( &(attrs[th]) );
1799    }
1800    (*p_count0)=0; for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
(*p_count0) = (*p_count0) + myargstruct_array[th].count0; }
1801    (*p_sum0)=0.0; for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) { (*p_sum0)
= (*p_sum0)    + myargstruct_array[th].sum0;    }
1802    (*p_count1)=0; for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
(*p_count1) = (*p_count1) + myargstruct_array[th].count1; }
1803    (*p_sum1)=0.0; for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) { (*p_sum1)
= (*p_sum1)    + myargstruct_array[th].sum1;    }
1804 }
1805 void compute_count_and_sum_uniform_time_do_parallel(int n_examples,int
j,int lo_index,int hi_index,int* p_count0,double* p_sum0,int*
p_count1,double* p_sum1) {
```

```
1806    int rc; int th; void* status;
1807    for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
1808       pthread_attr_init(&(attrs[th]));
1809       pthread_attr_setdetachstate(&(attrs[th]),PTHREAD_CREATE_JOINABLE);
1810
set_indices_in_myargstruct_array(th,lo_index,hi_index,COMPUTATION_TO_PERFORM_
IS_COMPUTE_COUNT_AND_SUM_UNIFORM_TIME,n_examples,j);
1811       rc = pthread_create(&(threads[th]),&(attrs[th]),worker,(void*)
(&(myargstruct_array[th])));
1812       if (rc){
1813          printf("ERROR; return code from pthread_create() is %d\n", rc);
1814          exit(-1);
1815       }
1816    }
1817    for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
1818       rc = pthread_join(threads[th], &status);
1819       if (rc) {
1820          printf("ERROR; return code from pthread_join() is %d\n", rc);
1821          exit(-1);
1822       }
1823    }
1824    for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
1825      pthread_attr_destroy( &(attrs[th]) );
1826    }
1827    (*p_count0)=0; for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
(*p_count0) = (*p_count0) + myargstruct_array[th].count0; }
1828    (*p_sum0)=0.0; for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) { (*p_sum0)
= (*p_sum0)    + myargstruct_array[th].sum0;    }
1829    (*p_count1)=0; for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
(*p_count1) = (*p_count1) + myargstruct_array[th].count1; }
1830    (*p_sum1)=0.0; for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) { (*p_sum1)
= (*p_sum1)    + myargstruct_array[th].sum1;    }
1831 }
1832 void update_prediction_and_error_single_bit_do_parallel(int
n_examples,int j,int lo_index,int hi_index) {
1833    int rc; int th; void* status;
1834    for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
1835       pthread_attr_init( &(attrs[th]) );
1836       pthread_attr_setdetachstate( &(attrs[th]),
PTHREAD_CREATE_JOINABLE);
1837
set_indices_in_myargstruct_array(th,lo_index,hi_index,COMPUTATION_TO_PERFORM_
IS_UPDATE_PREDICTION_AND_ERROR_SINGLE_BIT,n_examples,j);
1838       rc = pthread_create(&(threads[th]),&(attrs[th]),worker,(void*)
(&(myargstruct_array[th])));
1839       if (rc){
1840          printf("ERROR; return code from pthread_create() is %d\n", rc);
1841          exit(-1);
1842       }
1843    }
1844    for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
1845       rc = pthread_join(threads[th], &status);
1846       if (rc) {
1847          printf("ERROR; return code from pthread_join() is %d\n", rc);
1848          exit(-1);
1849       }
1850    }
```

```
1851   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
1852     pthread_attr_destroy( &(attrs[th]) );
1853   }
1854 }
1855 void update_prediction_and_error_all_bits_do_parallel(int n_examples,int
lo_index,int hi_index) {
1856   int rc; int th; void* status;
1857   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
1858      pthread_attr_init( &(attrs[th]) );
1859      pthread_attr_setdetachstate( &(attrs[th]),
PTHREAD_CREATE_JOINABLE);
1860
set_indices_in_myargstruct_array(th,lo_index,hi_index,COMPUTATION_TO_PERFORM_
IS_COMPUTE_PREDICTION_AND_ERROR_ALL_BITS,n_examples,-1);
1861      rc = pthread_create(&(threads[th]),&(attrs[th]),worker,(void*)
(&(myargstruct_array[th])));
1862      if (rc){
1863         printf("ERROR; return code from pthread_create() is %d\n", rc);
1864         exit(-1);
1865      }
1866   }
1867   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
1868      rc = pthread_join(threads[th], &status);
1869      if (rc) {
1870         printf("ERROR; return code from pthread_join() is %d\n", rc);
1871         exit(-1);
1872      }
1873   }
1874   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
1875     pthread_attr_destroy( &(attrs[th]) );
1876   }
1877 }
1878
1879 void compute_weight_for_regression_with_affine_function_single_bit(int
samping_method_for_regression_with_affine_function,int n_examples,int j,int
lo_index,int hi_index) {
1880   int count0; double sum0; int count1; double sum1;
1881   if
(samping_method_for_regression_with_affine_function==SAMPLING_METHOD_FOR_REGR
ESSION_WITH_AFFINE_FUNCTION_UNIFORM) {
1882
compute_count_and_sum_uniform_do_parallel(n_examples,j,lo_index,hi_index,&cou
nt0,&sum0,&count1,&sum1);
1883     if ((0<count0) && (0<count1)) {
1884       weights_for_regression_with_affine_function[j] = (-
1.0)*(sum1/count1 - sum0/count0)/2.0;
1885     } else {
1886       weights_for_regression_with_affine_function[j] = 0.0;
1887     }
1888   } else {
1889     if
(samping_method_for_regression_with_affine_function==SAMPLING_METHOD_FOR_REGR
ESSION_WITH_AFFINE_FUNCTION_UNIFORM_TIME) {
1890
compute_count_and_sum_uniform_time_do_parallel(n_examples,j,lo_index,hi_index
,&count0,&sum0,&count1,&sum1);
1891       if ((0<count0) && (0<count1)) {
```

```
1892          weights_for_regression_with_affine_function[j] = (-
1.0)*(sum1/count1 - sum0/count0)/2.0;
1893        } else {
1894          weights_for_regression_with_affine_function[j] = 0.0;
1895        }
1896      } else {
1897        printf("Error in
compute_weights_for_regression_with_affine_function weight_single_bit\n");
exit(-1);
1898      }
1899    }
1900 }
1901
1902 void do_regression_with_affine_function(int
samping_method_for_regression_with_affine_function,int n_examples) {
1903    double mint; double maxt; double averaget; int row_in_dataset; int j;
1904    printf("Starting do_regression_with_affine_function\n");
fflush(stdout);
1905    get_min_max_average_execution_time(&mint,&maxt,&averaget);
1906    for (row_in_dataset=0;row_in_dataset<n_examples;row_in_dataset++) {
1907      manyruns[row_in_dataset].prediction = averaget;
1908      manyruns[row_in_dataset].error = manyruns[row_in_dataset].prediction
- manyruns[row_in_dataset].t;
1909    }
1910    for (j=0;j<inputsizeinnumberofbits;j++) {
1911      if (j%1000==0) { printf("In write_initial_w0_to_file. %d\n",j); }
1912
compute_weight_for_regression_with_affine_function_single_bit(samping_method_
for_regression_with_affine_function,n_examples,j,0,n_examples-1);
1913
update_prediction_and_error_single_bit_do_parallel(n_examples,j,0,n_examples-
1);
1914    }
1915    printf("Finished do_regression_with_affine_function\n");
fflush(stdout);
1916 }
1917
1918 void write_weights_of_regression_with_affine_function_to_file(int
samping_method_for_regression_with_affine_function,char* fn_prefix) {
1919    FILE* f; char fn_to_use[2000];
1920
sprintf(fn_to_use,"%s_%d_weights_of_regression_with_affine_function_to_file.d
at",fn_prefix,samping_method_for_regression_with_affine_function);
1921    f = fopen(fn_to_use, "wb" ); if (f==NULL) { printf("Error in
write_weights_of_regression_with_affine_function_to_file\n"); exit(-1); }
1922
fwrite(weights_for_regression_with_affine_function,sizeof(double),inputsizein
numberofbits,f);
1923    fclose( f);
1924 }
1925 void read_weights_of_regression_with_affine_function_from_file(int
samping_method_for_regression_with_affine_function,char* fn_prefix) {
1926    FILE* f; char fn_to_use[2000];
1927
sprintf(fn_to_use,"%s_%d_weights_of_regression_with_affine_function_to_file.d
at",fn_prefix,samping_method_for_regression_with_affine_function);
```

```
1928   f = fopen(fn_to_use, "rb" ); if (f==NULL) { printf("Error in
read_weights_of_regression_with_affine_function_from_file\n"); exit(-1); }
1929
fread(weights_for_regression_with_affine_function,sizeof(double),inputsizeinn
umberofbits,f);
1930   fclose( f);
1931 }
1932
1933 void
compute_prediction_and_error_for_all_examples_coefficients_from_learned_affin
e_function(int n_examples) {
1934
update_prediction_and_error_all_bits_do_parallel(n_examples,0,n_examples-1);
1935 }
1936
1937 void
compute_statistics_based_on_regression_with_affine_function_based_on_all_exam
ples(int n_examples) {
1938   int row_in_dataset;
1939   row_in_dataset=0;
1940   regression_with_affine_function_min_prediction
= manyruns[row_in_dataset].prediction;
1941   regression_with_affine_function_max_prediction
= manyruns[row_in_dataset].prediction;
1942   regression_with_affine_function_max_abs_error
= fabs(manyruns[row_in_dataset].error);
1943   regression_with_affine_function_index_with_max_abs_error
= row_in_dataset;
1944
regression_with_affine_function_prediction_of_example_with_max_abs_error =
manyruns[row_in_dataset].prediction;
1945   regression_with_affine_function_t_of_example_with_max_abs_error
= manyruns[row_in_dataset].t;
1946   regression_with_affine_function_average_abs_error
= fabs(manyruns[row_in_dataset].error);
1947   for (row_in_dataset=1;row_in_dataset<n_examples;row_in_dataset++) {
1948     regression_with_affine_function_min_prediction    =
mindouble(regression_with_affine_function_min_prediction,manyruns[row_in_data
set].prediction);
1949     regression_with_affine_function_max_prediction    =
maxdouble(regression_with_affine_function_max_prediction,manyruns[row_in_data
set].prediction);
1950     if
(regression_with_affine_function_max_abs_error<fabs(manyruns[row_in_dataset].
error)) {
1951       regression_with_affine_function_max_abs_error
= fabs(manyruns[row_in_dataset].error);
1952       regression_with_affine_function_index_with_max_abs_error
= row_in_dataset;
1953
regression_with_affine_function_prediction_of_example_with_max_abs_error =
manyruns[row_in_dataset].prediction;
1954       regression_with_affine_function_t_of_example_with_max_abs_error
= manyruns[row_in_dataset].t;
1955     }
```

```
1956      regression_with_affine_function_average_abs_error =
regression_with_affine_function_average_abs_error +
fabs(manyruns[row_in_dataset].error);
1957    }
1958    regression_with_affine_function_average_abs_error =
regression_with_affine_function_average_abs_error / n_examples;
1959  }
1960
1961  void
compute_statistics_based_on_regression_with_affine_function_based_on_uniform_
time(int n_examples) {
1962    int iterator; int row_in_dataset; int niterations;
1963    regression_with_affine_function_average_abs_error_uniform_time = 0.0;
1964    iterator=0;
1965    niterations = n_examples / 10; // ideally we should not divide by 10
but it makes it run faster at the expense of slightly worse estimation
1966    while (iterator<niterations) {
1967      row_in_dataset =
obtain_row_using_uniform_sampling_over_execution_time(n_examples);
1968      regression_with_affine_function_average_abs_error_uniform_time =
regression_with_affine_function_average_abs_error_uniform_time +
fabs(manyruns[row_in_dataset].error);
1969      iterator++;
1970    }
1971    regression_with_affine_function_average_abs_error_uniform_time =
regression_with_affine_function_average_abs_error_uniform_time / niterations;
1972  }
1973
1974
1975  void compute_statistics_based_on_regression_with_affine_function(int
n_examples) {
1976
compute_statistics_based_on_regression_with_affine_function_based_on_all_exam
ples(n_examples);
1977
compute_statistics_based_on_regression_with_affine_function_based_on_uniform_
time(n_examples);
1978  }
1979
1980  void write_initial_w0_to_file(int n_examples,char* fn_prefix) {
1981    double mint; double maxt; double averaget; int row_in_dataset; int j;
double* temp_w;
1982    FILE* f; char fn_to_use[2000];
1983    sprintf(fn_to_use,"%s_initial_w0.txt",fn_prefix);
1984    f = fopen(fn_to_use, "w" ); if (f==NULL) { printf("Error in
write_initial_w0_to_file\n"); exit(-1); }
1985    for (j=0;j<inputsizeinnumberofbits;j++) {
1986
fprintf(f,"%15.12lf\n",weights_for_regression_with_affine_function[j]);
1987    }
1988    fclose( f);
1989    printf("Finished write_initial_w0_to_file\n"); fflush(stdout);
1990  }
1991
1992  void write_single_double_to_file(char* fn_prefix,char*
extra_part_of_fn,double t) {
1993    char fn_to_use[2000]; FILE* f;
```

```
1994   sprintf(fn_to_use,"%s_%s",fn_prefix,extra_part_of_fn);
1995   f = fopen(fn_to_use, "w" ); if (f==NULL) { printf("Error
write_single_double_to_file %s\n",fn_to_use); exit(-1); }
1996   fprintf(f,"%21.18lf\n",t);
1997   fclose( f);
1998 }
1999 void write_single_int_to_file(char* fn_prefix,char* extra_part_of_fn,int
t) {
2000   char fn_to_use[2000]; FILE* f;
2001   sprintf(fn_to_use,"%s_%s",fn_prefix,extra_part_of_fn);
2002   f = fopen(fn_to_use, "w" ); if (f==NULL) { printf("Error
write_single_int_to_file %s\n",fn_to_use); exit(-1); }
2003   fprintf(f,"%d\n",t);
2004   fclose( f);
2005 }
2006
2007 // this assumes that
compute_statistics_based_on_regression_with_affine_function has already been
called before
2008 void
print_statistics_on_prediction_and_error_of_training_using_affine_function(ch
ar* fn_prefix,int n_examples) {
2009
write_single_double_to_file(fn_prefix,"regression_with_affine_function_min_pr
ediction.txt",regression_with_affine_function_min_prediction);
2010
write_single_double_to_file(fn_prefix,"regression_with_affine_function_max_pr
ediction.txt",regression_with_affine_function_max_prediction);
2011
write_single_double_to_file(fn_prefix,"regression_with_affine_function_max_ab
s_error.txt",regression_with_affine_function_max_abs_error);
2012
write_single_double_to_file(fn_prefix,"regression_with_affine_function_averag
e_abs_error.txt",regression_with_affine_function_average_abs_error);
2013
write_single_double_to_file(fn_prefix,"regression_with_affine_function_averag
e_abs_error_uniform_time.txt",regression_with_affine_function_average_abs_err
or_uniform_time);
2014 }
2015
2016 double compute_max_abs_contribution_from_w0_coefficients(char*
fn_prefix) {
2017   double sum; double v; char fn_to_use[2000]; FILE* f; int j;
2018   sum = 0.0;
2019   sprintf(fn_to_use,"%s_initial_w0.txt",fn_prefix);
2020   f = fopen(fn_to_use, "r" ); if (f==NULL) { printf("Error in
compute_max_abs_contribution_from_w0_coefficients\n"); exit(-1); }
2021   for (j=0;j<inputsizeinnumberofbits;j++) {
2022     fscanf(f,"%lf", &v);
2023     sum = sum + fabs(v);
2024   }
2025   fclose( f);
2026   return sum;
2027 }
2028
```

```
2029 void
make_sure_span_is_not_greater_than_max_abs_contribution_from_w0_coefficients(
double* p_span, char* fn_prefix) {
2030   double max_abs_contribution_from_w0_coefficients;
2031   max_abs_contribution_from_w0_coefficients =
compute_max_abs_contribution_from_w0_coefficients(fn_prefix);
2032   if ((*p_span)>max_abs_contribution_from_w0_coefficients) {
2033     (*p_span)=max_abs_contribution_from_w0_coefficients;
2034   }
2035 }
2036
2037 double stored_weightconst_first_layer = -1.0;
2038 double stored_weightconst_second_layer = -1.0;
2039
2040 double stored_weightconst_to_be_used_for_w5_and_w6 = -1.0;
2041
2042 double stored_initialization_of_w5[HIDDEN_DIM];
2043 double stored_initialization_of_w6[HIDDEN_DIM];
2044
2045 double compute_contribution_single_run_compute_one_term(int*
p_temp_int_array,double attempted_weightconst_first_layer) {
2046   double sum; int j; int b; double w;
2047   sum = 0.0;
2048   for (j=0;j<inputsizeinnumberofbits;j++) {
2049     if (isbitonein_bitposition_in_integerarray(p_temp_int_array,j)) {
2050       b = 1;
2051     } else {
2052       b = 0;
2053     }
2054     w = attempted_weightconst_first_layer*(2*drand48()-1);
2055     sum = sum + w * (2*b-1);
2056   }
2057   b = 1;
2058   w = attempted_weightconst_first_layer*(2*drand48()-1);
2059   sum = sum + w * (2*b-1);
2060   return sum;
2061 }
2062
2063 void compute_contribution_single_run(double
attempted_weightconst_first_layer, double attempted_weightconst_second_layer,
double* p_max_abs_contribution_from_first_layer, double* p_sum_contribution)
{
2064   int* p_temp_int_array; double max_abs_contribution_from_first_layer;
double sum_contribution; int k; double term1; double term2; double w;
2065   p_temp_int_array =
malloc(sizeof(int)*getinputtoprogram_size_in_number_of_ints()); if
(p_temp_int_array==NULL) { printf("Memory allocation failure in
compute_contribution_single_run."); exit(-1); }
2066   fill_int_array_with_random_bits(p_temp_int_array);
2067   max_abs_contribution_from_first_layer = 0.0;
2068   sum_contribution = 0.0;
2069   for (k=0;k<HIDDEN_DIM;k++) {
2070     term1 =
compute_contribution_single_run_compute_one_term(p_temp_int_array,attempted_w
eightconst_first_layer);
```

```
2071      term2 =
compute_contribution_single_run_compute_one_term(p_temp_int_array,attempted_w
eightconst_first_layer);
2072      if
(max_abs_contribution_from_first_layer<fabs(mindouble(term1,term2))) {
max_abs_contribution_from_first_layer=fabs(mindouble(term1,term2)); }
2073      w = attempted_weightconst_second_layer*(2*drand48()-1);
2074      sum_contribution = sum_contribution + mindouble(term1,term2) * w;
2075    }
2076    for (k=0;k<HIDDEN_DIM;k++) {
2077      term1 =
compute_contribution_single_run_compute_one_term(p_temp_int_array,attempted_w
eightconst_first_layer);
2078      term2 =
compute_contribution_single_run_compute_one_term(p_temp_int_array,attempted_w
eightconst_first_layer);
2079      if
(max_abs_contribution_from_first_layer<fabs(maxdouble(term1,term2))) {
max_abs_contribution_from_first_layer=fabs(maxdouble(term1,term2)); }
2080      w = attempted_weightconst_second_layer*(2*drand48()-1);
2081      sum_contribution = sum_contribution + maxdouble(term1,term2) * w;
2082    }
2083    free(p_temp_int_array);
2084    *p_max_abs_contribution_from_first_layer =
max_abs_contribution_from_first_layer;
2085    *p_sum_contribution = sum_contribution;
2086  }
2087
2088  double find_attempted_weightconst_first_layer() {
2089    double lb_attempted_weightconst_first_layer;
2090    double ub_attempted_weightconst_first_layer;
2091    double attempted_weightconst_first_layer;
2092    int iterator;
2093    double max_abs_contribution_from_first_layer;
2094    double contribution;
2095    lb_attempted_weightconst_first_layer =    0.0;
2096    ub_attempted_weightconst_first_layer = 1000.0;
2097    for (iterator=0;iterator<40;iterator++) {
2098      attempted_weightconst_first_layer =
(lb_attempted_weightconst_first_layer +
ub_attempted_weightconst_first_layer)/2.0;
2099
compute_contribution_single_run(attempted_weightconst_first_layer,1.0,&max_ab
s_contribution_from_first_layer,&contribution);
2100      if (max_abs_contribution_from_first_layer<=1.0) {
2101        lb_attempted_weightconst_first_layer =
attempted_weightconst_first_layer;
2102      } else {
2103        ub_attempted_weightconst_first_layer =
attempted_weightconst_first_layer;
2104      }
2105    }
2106    return lb_attempted_weightconst_first_layer;
2107  }
2108  double find_attempted_weightconst_second_layer(double
weightconst_first_layer,double span) {
2109    double lb_attempted_weightconst_second_layer;
```

```
2110   double ub_attempted_weightconst_second_layer;
2111   double attempted_weightconst_second_layer;
2112   int iterator;
2113   double max_abs_contribution_from_first_layer;
2114   double contribution;
2115   lb_attempted_weightconst_second_layer =    0.0;
2116   ub_attempted_weightconst_second_layer = 1000.0;
2117   for (iterator=0;iterator<40;iterator++) {
2118     attempted_weightconst_second_layer =
(lb_attempted_weightconst_second_layer +
ub_attempted_weightconst_second_layer)/2.0;
2119
compute_contribution_single_run(weightconst_first_layer,attempted_weightconst
_second_layer,&max_abs_contribution_from_first_layer,&contribution);
2120     if (fabs(contribution)<=span) {
2121       lb_attempted_weightconst_second_layer =
attempted_weightconst_second_layer;
2122     } else {
2123       ub_attempted_weightconst_second_layer =
attempted_weightconst_second_layer;
2124     }
2125   }
2126   return lb_attempted_weightconst_second_layer;
2127 }
2128
2129 void use_simulation_to_compute_weightconst_and_store_it(char* fn_prefix)
{
2130   double mint; double maxt; double averaget; double span; double
weightconst_first_layer; double weightconst_second_layer; double
max_abs_contribution_from_w0_coefficients; int iterator;
2131   get_min_max_average_execution_time(&mint,&maxt,&averaget);
2132   span = (maxt-mint) / 4.0;
2133   printf("span = %lf\n", span);
2134   weightconst_first_layer = find_attempted_weightconst_first_layer();
2135   for (iterator=0;iterator<10;iterator++) {
2136     weightconst_first_layer =
mindouble(weightconst_first_layer,find_attempted_weightconst_first_layer());
2137   }
2138   printf("After find_attempted_weightconst_first_layer.
weightconst_first_layer = %lf\n", weightconst_first_layer);
2139   weightconst_second_layer =
find_attempted_weightconst_second_layer(weightconst_first_layer,span);
2140   for (iterator=0;iterator<10;iterator++) {
2141     weightconst_second_layer =
mindouble(weightconst_second_layer,find_attempted_weightconst_second_layer(we
ightconst_first_layer,span));
2142   }
2143   printf("After find_attempted_weightconst_second_layer.
weightconst_second_layer = %lf\n", weightconst_second_layer);
2144   stored_weightconst_first_layer = weightconst_first_layer;
2145   stored_weightconst_second_layer = weightconst_second_layer;
2146 }
2147
2148 double find_attempted_weightconst_to_be_used_for_w5_and_w6(double
target_atmost_contrbution) {
2149   double lb_attempted_weightconst_to_be_used_for_w5_and_w6;
2150   double ub_attempted_weightconst_to_be_used_for_w5_and_w6;
```

```
2151   double attempted_weightconst_to_be_used_for_w5_and_w6;
2152   int iterator; int k; double temp_b; double temp_w;
2153   double contribution;
2154   lb_attempted_weightconst_to_be_used_for_w5_and_w6 =    0.0;
2155   ub_attempted_weightconst_to_be_used_for_w5_and_w6 = 1000.0;
2156   for (iterator=0;iterator<40;iterator++) {
2157     attempted_weightconst_to_be_used_for_w5_and_w6 =
(lb_attempted_weightconst_to_be_used_for_w5_and_w6 +
ub_attempted_weightconst_to_be_used_for_w5_and_w6)/2.0;
2158     contribution = 0.0;
2159     for (k=0;k<2*HIDDEN_DIM;k++) {
2160       temp_b = (2*drand48()-1);
2161       temp_w =
attempted_weightconst_to_be_used_for_w5_and_w6*(2*drand48()-1);
2162       contribution = contribution + temp_b*temp_w;
2163     }
2164     if (fabs(contribution)<=target_atmost_contrbution) {
2165       lb_attempted_weightconst_to_be_used_for_w5_and_w6 =
attempted_weightconst_to_be_used_for_w5_and_w6;
2166     } else {
2167       ub_attempted_weightconst_to_be_used_for_w5_and_w6 =
attempted_weightconst_to_be_used_for_w5_and_w6;
2168     }
2169   }
2170   return lb_attempted_weightconst_to_be_used_for_w5_and_w6;
2171 }
2172
2173 void
use_simulation_to_compute_weightconst_to_be_used_for_w5_and_w6_and_store_it(c
har* fn_prefix) {
2174   double weightconst_to_be_used_for_w5_and_w6; int iterator;
2175   weightconst_to_be_used_for_w5_and_w6 =
find_attempted_weightconst_to_be_used_for_w5_and_w6(regression_with_affine_fu
nction_average_abs_error_uniform_time);
2176   for (iterator=0;iterator<10;iterator++) {
2177     weightconst_to_be_used_for_w5_and_w6 =
mindouble(weightconst_to_be_used_for_w5_and_w6,
2178
find_attempted_weightconst_to_be_used_for_w5_and_w6(regression_with_affine_fu
nction_average_abs_error_uniform_time));
2179   }
2180   stored_weightconst_to_be_used_for_w5_and_w6 =
weightconst_to_be_used_for_w5_and_w6;
2181 }
2182
2183 double stored_initialization_of_w5[HIDDEN_DIM];
2184 double stored_initialization_of_w6[HIDDEN_DIM];
2185
2186 GRBenv* env = NULL;
2187 GRBmodel* model = NULL;
2188 int error = 0;
2189 double* sol;
2190 int nelements_of_constr;
2191 int* ind;
2192 double* val;
2193 int nelements_of_obj;
2194 double* obj;
```

```
2195 double* lb;
2196 double* ub;
2197 char* vtype;
2198 int optimstatus;
2199 double objval;
2200 double rhs;
2201
2202 void dothememoryallocation_for_Gurobi_stuff() {
2203   sol   = (double*) malloc( nelements_of_obj * sizeof(double)); if
(sol==NULL)   { fprintf(stderr,"malloc failure in
dothememoryallocation_for_Gurobi_stuff: sol\n");   exit(-1); }
2204   ind   = (int*)    malloc( nelements_of_obj * sizeof(int));    if
(ind==NULL)   { fprintf(stderr,"malloc failure in
dothememoryallocation_for_Gurobi_stuff: ind\n");   exit(-1); }
2205   val   = (double*) malloc( nelements_of_obj * sizeof(double)); if
(val==NULL)   { fprintf(stderr,"malloc failure in
dothememoryallocation_for_Gurobi_stuff: val\n");   exit(-1); }
2206   obj   = (double*) malloc( nelements_of_obj * sizeof(double)); if
(obj==NULL)   { fprintf(stderr,"malloc failure in
dothememoryallocation_for_Gurobi_stuff: obj\n");   exit(-1); }
2207   lb    = (double*) malloc( nelements_of_obj * sizeof(double)); if
(lb==NULL)    { fprintf(stderr,"malloc failure in
dothememoryallocation_for_Gurobi_stuff: lb\n");   exit(-1); }
2208   ub    = (double*) malloc( nelements_of_obj * sizeof(double)); if
(ub==NULL)    { fprintf(stderr,"malloc failure in
dothememoryallocation_for_Gurobi_stuff: ub\n");   exit(-1); }
2209   vtype = (char*)   malloc( nelements_of_obj * sizeof(char));   if
(vtype==NULL) { fprintf(stderr,"malloc failure in
dothememoryallocation_for_Gurobi_stuff: vtype\n"); exit(-1); }
2210 }
2211
2212 void freethememory_for_Gurobi_stuff() {
2213   free( sol);
2214   free( ind);
2215   free( val);
2216   free( obj);
2217   free( lb);
2218   free( ub);
2219   free( vtype);
2220 }
2221
2222 void
fillobj_for_objective_function_compute_initialization_for_w5_and_w6() {
2223   int j;
2224   for (j=0;j<2*HIDDEN_DIM;j++) {
2225     obj[j]   = 0.0;
2226     lb[j]    = -GRB_INFINITY;
2227     ub[j]    = GRB_INFINITY;
2228     vtype[j] = GRB_CONTINUOUS;
2229   }
2230   j=2*HIDDEN_DIM;
2231   obj[j]   = 1.0;
2232   lb[j]    = -GRB_INFINITY;
2233   ub[j]    = GRB_INFINITY;
2234   vtype[j] = GRB_CONTINUOUS;
2235 }
2236
```

```
2237 void setup_datastructures_for_specific_w5_and_w6_leq_constraint(int
constraint_index) {
2238   int k;
2239   nelements_of_constr = 0;
2240   for (k=0;k<HIDDEN_DIM;k++) {
2241     ind[nelements_of_constr] = k;
2242     val[nelements_of_constr] =
array_with_data_for_w5_and_w6_leq_constraints[constraint_index].w5[k];
2243     nelements_of_constr = nelements_of_constr + 1;
2244   }
2245   for (k=0;k<HIDDEN_DIM;k++) {
2246     ind[nelements_of_constr] = HIDDEN_DIM+k;
2247     val[nelements_of_constr] =
array_with_data_for_w5_and_w6_leq_constraints[constraint_index].w6[k];
2248     nelements_of_constr = nelements_of_constr + 1;
2249   }
2250   ind[nelements_of_constr] = 2*HIDDEN_DIM;
2251   val[nelements_of_constr] = -1.0;
2252   nelements_of_constr = nelements_of_constr + 1;
2253   rhs =
array_with_data_for_w5_and_w6_leq_constraints[constraint_index].rhs;
2254 }
2255
2256 void setup_datastructures_for_specific_w5_and_w6_flip_from_leq_to_geq()
{
2257   val[2*HIDDEN_DIM] = val[2*HIDDEN_DIM] * (-1.0);
2258 }
2259
2260 void read_initial_wx_from_file(int n_examples,char* fn_prefix,int x) {
2261   int j; int k; FILE* f; char fn_to_use[2000];
2262   sprintf(fn_to_use,"%s_initial_w%d.txt",fn_prefix,x);
2263   f = fopen(fn_to_use, "r" ); if (f==NULL) { printf("Error in
read_initial_wx_from_file\n"); exit(-1); }
2264   for (j=0;j<inputsizeinnumberofbits;j++) {
2265     for (k=0;k<HIDDEN_DIM;k++) {
2266       if (x==1) {
2267         fscanf(f,"%lf", &(p_w1_value[j*HIDDEN_DIM+k]));
2268       } else {
2269         if (x==2) {
2270           fscanf(f,"%lf", &(p_w2_value[j*HIDDEN_DIM+k]));
2271         } else {
2272           if (x==3) {
2273             fscanf(f,"%lf", &(p_w3_value[j*HIDDEN_DIM+k]));
2274           } else {
2275             if (x==4) {
2276               fscanf(f,"%lf", &(p_w4_value[j*HIDDEN_DIM+k]));
2277             } else {
2278               printf("read_initial_wx_from_file\n"); exit(-1);
2279             }
2280           }
2281         }
2282       }
2283     }
2284   }
2285   fclose( f);
2286 }
2287
```

```
2288 void read_initial_bx_from_file(int n_examples,char* fn_prefix,int x) {
2289   int k; FILE* f; char fn_to_use[2000];
2290   sprintf(fn_to_use,"%s_initial_b%d.txt",fn_prefix,x);
2291   f = fopen(fn_to_use, "r" ); if (f==NULL) { printf("Error in
read_initial_bx_from_file\n"); exit(-1); }
2292   for (k=0;k<HIDDEN_DIM;k++) {
2293     if (x==1) {
2294       fscanf(f,"%lf", &(p_b1_value[k]));
2295     } else {
2296       if (x==2) {
2297         fscanf(f,"%lf", &(p_b2_value[k]));
2298       } else {
2299         if (x==3) {
2300           fscanf(f,"%lf", &(p_b3_value[k]));
2301         } else {
2302           if (x==4) {
2303             fscanf(f,"%lf", &(p_b4_value[k]));
2304           } else {
2305             printf("read_initial_bx_from_file\n"); exit(-1);
2306           }
2307         }
2308       }
2309     }
2310   }
2311   fclose( f);
2312 }
2313
2314 void read_initial_w1_from_file(int n_examples,char* fn_prefix) {
read_initial_wx_from_file(n_examples,fn_prefix,1); }
2315 void read_initial_w2_from_file(int n_examples,char* fn_prefix) {
read_initial_wx_from_file(n_examples,fn_prefix,2); }
2316 void read_initial_w3_from_file(int n_examples,char* fn_prefix) {
read_initial_wx_from_file(n_examples,fn_prefix,3); }
2317 void read_initial_w4_from_file(int n_examples,char* fn_prefix) {
read_initial_wx_from_file(n_examples,fn_prefix,4); }
2318 void read_initial_b1_from_file(int n_examples,char* fn_prefix) {
read_initial_bx_from_file(n_examples,fn_prefix,1); }
2319 void read_initial_b2_from_file(int n_examples,char* fn_prefix) {
read_initial_bx_from_file(n_examples,fn_prefix,2); }
2320 void read_initial_b3_from_file(int n_examples,char* fn_prefix) {
read_initial_bx_from_file(n_examples,fn_prefix,3); }
2321 void read_initial_b4_from_file(int n_examples,char* fn_prefix) {
read_initial_bx_from_file(n_examples,fn_prefix,4); }
2322
2323 // We have 2*HIDDEN_DIM+1 variables
2324 // for index 0          ..  HIDDEN_DIM-1, we have the w5 variables
2325 // for index HIDDEN_DIM.. 2*HIDDEN_DIM-1, we have the w6 variables
2326 // for index 2*HIDDEN_DIM,                we have the maxabserror
variable (that we want to minimize)
2327
2328 // generating the LP instance takes  20 minutes with
"constraint_iterator<100000"
2329 // generating the LP instance takes 100 minutes with
"constraint_iterator<300000"
2330 // generating the LP instance is expected to take 200 minutes with
"constraint_iterator<600000"
2331
```

```
2332 void compute_initialization_for_w5_and_w6_by_solving_MILP(char*
fn_prefix,int n_examples) {
2333   int j; int k; int index1; int index2; int selected_index;
2334   char gurobi_initialization_w5_w6_log_file[200];
2335   char gurobi_initialization_w5_w6_lp_file[200];
2336   char gurobi_initialization_w5_w6_sol_file[200];
2337   int row_in_dataset;
2338   int constraint_index;
2339   int* array_of_row_indices;
2340   printf("Start
compute_initialization_for_w5_and_w6_by_solving_MILP\n"); fflush(stdout);
2341   read_initial_w1_from_file(n_examples,fn_prefix);
2342   read_initial_w2_from_file(n_examples,fn_prefix);
2343   read_initial_w3_from_file(n_examples,fn_prefix);
2344   read_initial_w4_from_file(n_examples,fn_prefix);
2345   read_initial_b1_from_file(n_examples,fn_prefix);
2346   read_initial_b2_from_file(n_examples,fn_prefix);
2347   read_initial_b3_from_file(n_examples,fn_prefix);
2348   read_initial_b4_from_file(n_examples,fn_prefix);
2349   printf("Done reading w1,w2,w3,w4,b1,b2,b3,b4\n"); fflush(stdout);
2350
2351
sprintf(gurobi_initialization_w5_w6_log_file,"%s_initialization_w5_w6.log",fn
_prefix);
2352   sprintf(gurobi_initialization_w5_w6_lp_file,
"%s_initialization_w5_w6.lp", fn_prefix);
2353
sprintf(gurobi_initialization_w5_w6_sol_file,"%s_initialization_w5_w6.sol",fn
_prefix);
2354   nelements_of_obj = 2*HIDDEN_DIM+1;
2355   dothememoryallocation_for_Gurobi_stuff();
2356   error = GRBloadenv(&env, gurobi_initialization_w5_w6_log_file);
2357   if (error) goto QUIT;
2358   error = GRBnewmodel(env, &model, "initialization_w5_w6", 0, NULL,
NULL, NULL, NULL, NULL);
2359   if (error) goto QUIT;
2360   fillobj_for_objective_function_compute_initialization_for_w5_and_w6();
2361   error = GRBaddvars( model, nelements_of_obj, 0, NULL, NULL, NULL, obj,
lb, ub, vtype, NULL);
2362   if (error) goto QUIT;
2363   error = GRBsetintattr( model, GRB_INT_ATTR_MODELSENSE, GRB_MINIMIZE);
2364   if (error) goto QUIT;
2365   error = GRBupdatemodel( model);
2366   if (error) goto QUIT;
2367   allocate_memory_array_of_indices();
2368   allocate_memory_array_with_data_for_w5_and_w6_leq_constraints();
2369   printf("Before fill_array_of_row_indices\n"); fflush(stdout);
2370   fill_array_of_row_indices(n_examples);
2371   printf("After fill_array_of_row_indices\n"); fflush(stdout);
2372   printf("Before fill_array_with_data_for_w5_and_w6_leq_constraints\n");
fflush(stdout);
2373   fill_array_with_data_for_w5_and_w6_leq_constraints();
2374   printf("After fill_array_with_data_for_w5_and_w6_leq_constraints\n");
fflush(stdout);
2375   for
(constraint_index=0;constraint_index<NUMBER_OF_CONSTRAINTS_FOR_MILP_INITIALIZ
ING_W5_AND_W6;constraint_index++) {
```

```
2376
setup_datastructures_for_specific_w5_and_w6_leq_constraint(constraint_index);
2377     error = GRBaddconstr( model, nelements_of_constr, ind, val,
GRB_LESS_EQUAL, rhs, NULL );
2378     setup_datastructures_for_specific_w5_and_w6_flip_from_leq_to_geq();
2379     error = GRBaddconstr( model, nelements_of_constr, ind, val,
GRB_GREATER_EQUAL, rhs, NULL );
2380   }
2381   free_memory_array_of_row_indices();
2382   free_memory_array_with_data_for_w5_and_w6_constraints();
2383   error = GRBupdatemodel( model);
2384   if (error) goto QUIT;
2385   error = GRBoptimize(model);
2386   if (error) goto QUIT;
2387   error = GRBupdatemodel( model);
2388   if (error) goto QUIT;
2389   error = GRBgetintattr( model, GRB_INT_ATTR_STATUS, &optimstatus);
2390   if (error) goto QUIT;
2391   if (optimstatus == GRB_OPTIMAL) {
2392     error = GRBwrite( model,gurobi_initialization_w5_w6_sol_file);
2393     if (error) goto QUIT;
2394     error = GRBgetdblattr( model, GRB_DBL_ATTR_OBJVAL, &objval);
2395     if (error) goto QUIT;
2396     error = GRBgetdblattrarray( model, GRB_DBL_ATTR_X, 0,
nelements_of_obj, sol);
2397     if (error) goto QUIT;
2398     for (k=0;k<HIDDEN_DIM;k++) {
2399       stored_initialization_of_w5[k] = sol[k];
2400     }
2401     for (k=0;k<HIDDEN_DIM;k++) {
2402       stored_initialization_of_w6[k] = sol[HIDDEN_DIM+k];
2403     }
2404   } else {
2405     printf("An error in
compute_initialization_for_w5_and_w6_by_solving_MILP. Did not get an
objective.\n"); fflush(stdout); exit(-1);
2406   }
2407 QUIT:
2408   if (error) { printf("ERROR: %s\n", GRBgeterrormsg(env)); exit(1); }
2409   GRBfreemodel(model);
2410   GRBfreeenv(env);
2411   freethememory_for_Gurobi_stuff();
2412   printf("Finish
compute_initialization_for_w5_and_w6_by_solving_MILP\n"); fflush(stdout);
2413 }
2414
2415 double get_weightconst_first_layer() {
2416   if (stored_weightconst_first_layer==-1.0) {
2417     printf("In get_weightconst_first_layer. This should not happen.\n");
exit(-1);
2418   } else {
2419     return stored_weightconst_first_layer;
2420   }
2421 }
2422 double get_weightconst_second_layer() {
2423   if (stored_weightconst_second_layer==-1.0) {
```

```
2424     printf("In get_weightconst_second_layer. This should not
happen.\n"); exit(-1);
2425   } else {
2426     return stored_weightconst_second_layer;
2427   }
2428 }
2429
2430 void write_initial_wx_to_file(int n_examples,char* fn_prefix,int x) {
2431   int j; int k; FILE* f; char fn_to_use[2000]; double v; double
weightconst;
2432   weightconst = get_weightconst_first_layer();
2433   sprintf(fn_to_use,"%s_initial_w%d.txt",fn_prefix,x);
2434   f = fopen(fn_to_use, "w" ); if (f==NULL) { printf("Error in
fill_proposed_initial_wx_to_file\n"); exit(-1); }
2435   for (j=0;j<inputsizeinnumberofbits;j++) {
2436     for (k=0;k<HIDDEN_DIM;k++) {
2437       v = weightconst * (2*drand48()-1.0);
2438       fprintf(f,"%15.12lf ", v);
2439     }
2440     fprintf(f,"\n");
2441   }
2442   fclose( f);
2443 }
2444 void write_initial_w1_to_file(int n_examples,char* fn_prefix) {
write_initial_wx_to_file(n_examples,fn_prefix,1); }
2445 void write_initial_w2_to_file(int n_examples,char* fn_prefix) {
write_initial_wx_to_file(n_examples,fn_prefix,2); }
2446 void write_initial_w3_to_file(int n_examples,char* fn_prefix) {
write_initial_wx_to_file(n_examples,fn_prefix,3); }
2447 void write_initial_w4_to_file(int n_examples,char* fn_prefix) {
write_initial_wx_to_file(n_examples,fn_prefix,4); }
2448
2449 void write_initial_bx_to_file(int n_examples,char* fn_prefix,int x) {
2450   int j; int k; FILE* f; char fn_to_use[2000]; double v; double
weightconst;
2451   weightconst = get_weightconst_first_layer();
2452   sprintf(fn_to_use,"%s_initial_b%d.txt",fn_prefix,x);
2453   f = fopen(fn_to_use, "w" ); if (f==NULL) { printf("Error in
fill_proposed_initial_bx_to_file\n"); exit(-1); }
2454   for (k=0;k<HIDDEN_DIM;k++) {
2455     v = weightconst * (2*drand48()-1.0);
2456     fprintf(f,"%15.12lf ", v);
2457   }
2458   fprintf(f,"\n");
2459   fclose( f);
2460 }
2461 void write_initial_b1_to_file(int n_examples,char* fn_prefix) {
write_initial_bx_to_file(n_examples,fn_prefix,1); }
2462 void write_initial_b2_to_file(int n_examples,char* fn_prefix) {
write_initial_bx_to_file(n_examples,fn_prefix,2); }
2463 void write_initial_b3_to_file(int n_examples,char* fn_prefix) {
write_initial_bx_to_file(n_examples,fn_prefix,3); }
2464 void write_initial_b4_to_file(int n_examples,char* fn_prefix) {
write_initial_bx_to_file(n_examples,fn_prefix,4); }
2465
2466 void write_initial_wy_to_file(int n_examples,char* fn_prefix,int y) {
```

```
2467   int j; int k; FILE* f; char fn_to_use[2000]; double v; // double
weightconst;
2468   sprintf(fn_to_use,"%s_initial_w%d.txt",fn_prefix,y);
2469   f = fopen(fn_to_use, "w" ); if (f==NULL) { printf("Error in
fill_proposed_initial_wy_to_file\n"); exit(-1); }
2470   for (k=0;k<HIDDEN_DIM;k++) {
2471     // v = weightconst * (2*drand48()-1.0);
2472     // v =
(regression_with_affine_function_average_abs_error/(2.0*HIDDEN_DIM)) *
(2*drand48()-1.0);
2473     // v =
(regression_with_affine_function_average_abs_error_uniform_time/(2.0*HIDDEN_D
IM)) * (2*drand48()-1.0);
2474     // v =
regression_with_affine_function_average_abs_error_uniform_time *
(2*drand48()-1.0);
2475     // v = stored_weightconst_to_be_used_for_w5_and_w6 * (2*drand48()-
1.0);
2476     if (y==5) {
2477       v = stored_initialization_of_w5[k];
2478     } else {
2479       if (y==6) {
2480         v = stored_initialization_of_w6[k];
2481       } else {
2482         printf("Error in write_initial_wy_to_file\n"); exit(-1);
2483       }
2484     }
2485     fprintf(f,"%15.12lf ", v);
2486     fprintf(f,"\n");
2487   }
2488   fclose( f);
2489 }
2490 void write_initial_w5_to_file(int n_examples,char* fn_prefix) {
write_initial_wy_to_file(n_examples,fn_prefix,5); }
2491 void write_initial_w6_to_file(int n_examples,char* fn_prefix) {
write_initial_wy_to_file(n_examples,fn_prefix,6); }
2492
2493 void write_initial_b56_to_file(int n_examples,char* fn_prefix) {
2494   int j; int k; FILE* f; char fn_to_use[2000]; double v;
2495   double mint; double maxt; double averaget;
2496   sprintf(fn_to_use,"%s_initial_b56.txt",fn_prefix);
2497   f = fopen(fn_to_use, "w" ); if (f==NULL) { printf("Error in
fill_proposed_initial_b56_to_file\n"); exit(-1); }
2498   get_min_max_average_execution_time(&mint,&maxt,&averaget);
2499   v = averaget;
2500   fprintf(f,"%15.12lf ", v);
2501   fclose( f);
2502 }
2503
2504 void create_python_file(char* fn_prefix) {
2505   double mint; double maxt; double averaget;
2506   double weightconst;
2507   FILE* f;
2508   char python_fn[2000];
2509   get_min_max_average_execution_time(&mint,&maxt,&averaget);
2510   weightconst = (maxt-mint) / ( 2*(HIDDEN_DIM*2)*get_input_dim()  );
2511   weightconst = sqrt( weightconst );
```

```
2512   sprintf(python_fn,"%s_model.py",fn_prefix);
2513   f = fopen(python_fn, "w" );
2514   fprintf(f,"import tensorflow as tf\n");
2515   fprintf(f,"import numpy as np\n");
2516   fprintf(f,"tf.compat.v1.disable_v2_behavior()\n");
2517   fprintf(f,"with tf.compat.v1.Session() as sess:\n");
2518   fprintf(f,"  input_dim  = %d\n", get_input_dim());
2519   fprintf(f,"  hidden_dim = %d\n", HIDDEN_DIM);
2520   fprintf(f,"  x                  =
tf.compat.v1.placeholder(dtype=tf.float64, shape=[None,input_dim],
name='input')\n");
2521   fprintf(f,"  y                  =
tf.compat.v1.placeholder(dtype=tf.float64, shape=[None,1],
name='target')\n");
2522   fprintf(f,"  np_w0 = np.loadtxt(\"%s_initial_w0.txt\")\n",fn_prefix);
2523   fprintf(f,"  np_w0 = np.reshape(np_w0,(input_dim, 1))\n");
2524   fprintf(f,"  w0                 = tf.Variable( shape=[input_dim,1],
initial_value = np_w0 , name='w0')\n");
2525   fprintf(f,"  np_w1 = np.loadtxt(\"%s_initial_w1.txt\")\n",fn_prefix);
2526   fprintf(f,"  np_w1 = np.reshape(np_w1,(input_dim, hidden_dim))\n");
2527   fprintf(f,"  w1                 = tf.Variable(
shape=[input_dim,hidden_dim], initial_value = np_w1 , name='w1')\n");
2528   fprintf(f,"  np_w2 = np.loadtxt(\"%s_initial_w2.txt\")\n",fn_prefix);
2529   fprintf(f,"  np_w2 = np.reshape(np_w2,(input_dim, hidden_dim))\n");
2530   fprintf(f,"  w2                 = tf.Variable(
shape=[input_dim,hidden_dim], initial_value = np_w2 , name='w2')\n");
2531   fprintf(f,"  np_w3 = np.loadtxt(\"%s_initial_w3.txt\")\n",fn_prefix);
2532   fprintf(f,"  np_w3 = np.reshape(np_w3,(input_dim, hidden_dim))\n");
2533   fprintf(f,"  w3                 = tf.Variable(
shape=[input_dim,hidden_dim], initial_value = np_w3 , name='w3')\n");
2534   fprintf(f,"  np_w4 = np.loadtxt(\"%s_initial_w4.txt\")\n",fn_prefix);
2535   fprintf(f,"  np_w4 = np.reshape(np_w4,(input_dim, hidden_dim))\n");
2536   fprintf(f,"  w4                 = tf.Variable(
shape=[input_dim,hidden_dim], initial_value = np_w4 , name='w4')\n");
2537   fprintf(f,"  np_b1 = np.loadtxt(\"%s_initial_b1.txt\")\n",fn_prefix);
2538   fprintf(f,"  np_b1 = np.reshape(np_b1,(1, hidden_dim))\n");
2539   fprintf(f,"  b1                 = tf.Variable( shape=[1,hidden_dim],
initial_value = np_b1 , name='b1')\n");
2540   fprintf(f,"  np_b2 = np.loadtxt(\"%s_initial_b2.txt\")\n",fn_prefix);
2541   fprintf(f,"  np_b2 = np.reshape(np_b2,(1, hidden_dim))\n");
2542   fprintf(f,"  b2                 = tf.Variable( shape=[1,hidden_dim],
initial_value = np_b2 , name='b2')\n");
2543   fprintf(f,"  np_b3 = np.loadtxt(\"%s_initial_b3.txt\")\n",fn_prefix);
2544   fprintf(f,"  np_b3 = np.reshape(np_b3,(1, hidden_dim))\n");
2545   fprintf(f,"  b3                 = tf.Variable( shape=[1,hidden_dim],
initial_value = np_b3 , name='b3')\n");
2546   fprintf(f,"  np_b4 = np.loadtxt(\"%s_initial_b4.txt\")\n",fn_prefix);
2547   fprintf(f,"  np_b4 = np.reshape(np_b4,(1, hidden_dim))\n");
2548   fprintf(f,"  b4                 = tf.Variable( shape=[1,hidden_dim],
initial_value = np_b4 , name='b4')\n");
2549   fprintf(f,"  zmin  = tf.minimum( tf.matmul(2*x-1,w1)+b1,
tf.matmul(2*x-1,w2)+b2)\n");
2550   fprintf(f,"  zmax  = tf.maximum( tf.matmul(2*x-1,w3)+b3,
tf.matmul(2*x-1,w4)+b4)\n");
2551   fprintf(f,"  np_w5 = np.loadtxt(\"%s_initial_w5.txt\")\n",fn_prefix);
2552   fprintf(f,"  np_w5 = np.reshape(np_w5,(hidden_dim, 1))\n");
```

```
2553   fprintf(f,"  w5                 = tf.Variable( shape=[hidden_dim,1],
initial_value = np_w5 , name='w5')\n");
2554   fprintf(f,"  np_w6 = np.loadtxt(\"%s_initial_w6.txt\")\n",fn_prefix);
2555   fprintf(f,"  np_w6 = np.reshape(np_w6,(hidden_dim, 1))\n");
2556   fprintf(f,"  w6                 = tf.Variable( shape=[hidden_dim,1],
initial_value = np_w6 , name='w6')\n");
2557   fprintf(f,"  np_b56 =
np.loadtxt(\"%s_initial_b56.txt\")\n",fn_prefix);
2558   fprintf(f,"  np_b56 = np.reshape(np_b56,(1, 1))\n");
2559   fprintf(f,"  b56                = tf.Variable( shape=[1,1],
initial_value = np_b56 , name='b56')\n");
2560   fprintf(f,"  y_  =
tf.compat.v1.identity(tf.matmul(zmin,w5)+tf.matmul(zmax,w6)+tf.matmul(2*x-
1,w0)+b56, name='output')\n");
2561
2562   // # Optimize loss
2563   fprintf(f,"  loss = tf.reduce_mean(tf.square(y_ - y),
name='loss')\n");
2564   // fprintf(f,"  optimizer =
tf.compat.v1.train.GradientDescentOptimizer(learning_rate=0.0001)\n"); //
This takes __ minutes. This cause the output ___, ___, ___.
2565   // fprintf(f,"  optimizer =
tf.compat.v1.train.AdamOptimizer(learning_rate=0.000010)\n");
2566   // fprintf(f,"  optimizer =
tf.compat.v1.train.AdamOptimizer(learning_rate=0.000001)\n");
2567   // fprintf(f,"  optimizer =
tf.compat.v1.train.AdamOptimizer(learning_rate=0.0000001)\n");
2568   // fprintf(f,"  optimizer =
tf.compat.v1.train.AdamOptimizer(learning_rate=0.00000001)\n");
2569   // fprintf(f,"  optimizer =
tf.compat.v1.train.AdamOptimizer(learning_rate=0.000000001)\n");
2570   // fprintf(f,"  optimizer =
tf.compat.v1.train.AdamOptimizer(learning_rate=0.0000000001)\n");
2571   // fprintf(f,"  optimizer = tf.compat.v1.train.AdamOptimizer()\n");
2572   // fprintf(f,"  optimizer =
tf.compat.v1.train.AdamOptimizer(learning_rate=0.000000000001)\n");
2573   // fprintf(f,"  optimizer =
tf.compat.v1.train.GradientDescentOptimizer(learning_rate=0.1)\n");
2574   // fprintf(f,"  optimizer =
tf.compat.v1.train.GradientDescentOptimizer(learning_rate=0.01)\n");
2575   // fprintf(f,"  optimizer =
tf.compat.v1.train.GradientDescentOptimizer(learning_rate=0.001)\n");
2576   // fprintf(f,"  optimizer =
tf.compat.v1.train.GradientDescentOptimizer(learning_rate=0.0001)\n");
2577   // fprintf(f,"  optimizer =
tf.compat.v1.train.GradientDescentOptimizer(learning_rate=0.00001)\n");
2578   // fprintf(f,"  optimizer =
tf.compat.v1.train.GradientDescentOptimizer(learning_rate=0.000001)\n");
2579   // fprintf(f,"  optimizer =
tf.compat.v1.train.GradientDescentOptimizer(learning_rate=0.00001)\n"); //
this learning rate seems to work best for GD
2580   // fprintf(f,"  optimizer =
tf.compat.v1.train.GradientDescentOptimizer(learning_rate=0.0001)\n");
2581   // fprintf(f,"  optimizer = tf.compat.v1.train.AdamOptimizer()\n"); //
this, with minibatchsize=128, causes oscillations
```

```
2582   // fprintf(f,"  optimizer =
tf.compat.v1.train.AdamOptimizer(amsgrad=True)\n"); // this still suffers
from oscillations
2583   // fprintf(f,"  optimizer =
tf.compat.v1.train.AdamOptimizer(epsilon=0.1)\n"); // this still suffers from
oscillations
2584   // fprintf(f,"  optimizer =
tf.compat.v1.train.AdamOptimizer(epsilon=1.0)\n"); // this still suffers from
oscillations
2585   // fprintf(f,"  optimizer =
tf.compat.v1.train.GradientDescentOptimizer(learning_rate=0.00001)\n"); //
this works OK in the sense that there is minor shoot and there is no
oscillations but unfortunately, learning stops although there is remaining
average training error
2586   // fprintf(f,"  optimizer =
tf.compat.v1.train.GradientDescentOptimizer(learning_rate=0.0001)\n"); //
this sort-of-works but it does not offer the best WCET estimate
2587   // fprintf(f,"  optimizer =
tf.compat.v1.train.GradientDescentOptimizer(learning_rate=0.00001)\n");
2588   // fprintf(f,"  optimizer =
tf.compat.v1.train.GradientDescentOptimizer(learning_rate=0.000001)\n");
2589   // fprintf(f,"  optimizer =
tf.compat.v1.train.GradientDescentOptimizer(learning_rate=0.00001)\n"); // I
set w5 and w6 properly and when used this learning rate and it works; no
oscillations
2590   // fprintf(f,"  optimizer =
tf.compat.v1.train.GradientDescentOptimizer(learning_rate=0.0001)\n"); // I
set w5 and w6 properly and when used this learning rate and it works; some
oscillations but the error gets lower, assuming minibatch size=8 and 50000
minibatches of training; perhaps, we should use minibatch size=128
2591   fprintf(f,"  optimizer =
tf.compat.v1.train.GradientDescentOptimizer(learning_rate=0.0001)\n");
2592
2593   fprintf(f,"  train_op = optimizer.minimize(loss, name='train')\n");
2594   fprintf(f,"  init = tf.compat.v1.global_variables_initializer()\n");
2595
2596   // Defaults for Adam: learning_rate=0.001, beta_1=0.9, beta_2=0.999,
epsilon=1e-07, amsgrad=False
2597
2598   // # tf.train.Saver.__init__ adds operations to the graph to save
2599   // # and restore variables.
2600   fprintf(f,"  saver_def =
tf.compat.v1.train.Saver().as_saver_def()\n");
2601
2602   // # Write the graph out to a file.
2603   fprintf(f,"  with open('%s_graph.pb', 'wb') as f:\n",fn_prefix);
2604   fprintf(f,"
f.write(tf.compat.v1.get_default_graph().as_graph_def().SerializeToString())\
n");
2605   fclose( f);
2606 }
2607
2608 void print_weights_to_file_single_number(char* fn_prefix_string,int
id,char* fn,double* p) {
2609   FILE* f; char fn_to_use[2000];
2610   sprintf(fn_to_use,"%s_%d_%s",fn_prefix_string,id,fn);
```

```
2611   f = fopen(fn_to_use, "w" ); if (f==NULL) { printf("Error in
print_weights_to_file_single_number\n"); exit(-1); }
2612   fprintf(f,"%21.18lf\n",p[0]);
2613   fclose(f);
2614 }
2615 void print_weights_to_file_vector(char* fn_prefix_string,int id,char*
fn,double* p,int n_elements) {
2616   FILE* f; int index; char fn_to_use[2000];
2617   sprintf(fn_to_use,"%s_%d_%s",fn_prefix_string,id,fn);
2618   f = fopen(fn_to_use, "w" ); if (f==NULL) { printf("Error in
print_weights_to_file_vector\n"); exit(-1); }
2619   for (index=0;index<n_elements;index++) {
2620     fprintf(f,"%21.18lf\n",p[index]);
2621   }
2622   fclose(f);
2623 }
2624 void print_weights_to_file_matrix(char* fn_prefix_string,int id,char*
fn,double* p,int n_rows,int n_columns) {
2625   FILE* f; int row; int column; char fn_to_use[2000];
2626   sprintf(fn_to_use,"%s_%d_%s",fn_prefix_string,id,fn);
2627   f = fopen(fn_to_use, "w" ); if (f==NULL) { printf("Error in
print_weights_to_file_matrix\n"); exit(-1); }
2628   for (row=0;row<n_rows;row++) {
2629     for (column=0;column<n_columns;column++) {
2630       fprintf(f,"%21.18lf\n",p[row*n_columns+column]);
2631     }
2632   }
2633   fclose(f);
2634 }
2635
2636 void read_weights_to_file_single_number(char* fn_prefix_string,int
id,char* fn,double* p) {
2637   FILE* f; char fn_to_use[2000];
2638   sprintf(fn_to_use,"%s_%d_%s",fn_prefix_string,id,fn);
2639   f = fopen(fn_to_use, "r" ); if (f==NULL) { printf("Error in
read_weights_to_file_single_number\n"); exit(-1); }
2640   fscanf(f,"%lf",&(p[0]));
2641   fclose(f);
2642 }
2643 void read_weights_to_file_vector(char* fn_prefix_string,int id,char*
fn,double* p,int n_elements) {
2644   FILE* f; int index; char fn_to_use[2000];
2645   sprintf(fn_to_use,"%s_%d_%s",fn_prefix_string,id,fn);
2646   f = fopen(fn_to_use, "r" ); if (f==NULL) { printf("Error in
read_weights_to_file_vector\n"); exit(-1); }
2647   for (index=0;index<n_elements;index++) {
2648     fscanf(f,"%lf",&(p[index]));
2649   }
2650   fclose(f);
2651 }
2652 void read_weights_to_file_matrix(char* fn_prefix_string,int id,char*
fn,double* p,int n_rows,int n_columns) {
2653   FILE* f; int row; int column; char fn_to_use[2000];
2654   sprintf(fn_to_use,"%s_%d_%s",fn_prefix_string,id,fn);
2655   f = fopen(fn_to_use, "r" ); if (f==NULL) { printf("Error in
read_weights_to_file_matrix\n"); exit(-1); }
2656   for (row=0;row<n_rows;row++) {
```

```
2657     for (column=0;column<n_columns;column++) {
2658       fscanf(f,"%lf",&(p[row*n_columns+column]));
2659     }
2660   }
2661   fclose(f);
2662 }
2663
2664 void print_w0_to_file(char*  fn_prefix_string,int id) {
print_weights_to_file_vector(fn_prefix_string,id,"w0.txt", p_w0_value,
inputsizeinnumberofbits); }
2665 void print_w1_to_file(char*  fn_prefix_string,int id) {
print_weights_to_file_matrix(fn_prefix_string,id,"w1.txt", p_w1_value,
inputsizeinnumberofbits,HIDDEN_DIM); }
2666 void print_w2_to_file(char*  fn_prefix_string,int id) {
print_weights_to_file_matrix(fn_prefix_string,id,"w2.txt", p_w2_value,
inputsizeinnumberofbits,HIDDEN_DIM); }
2667 void print_w3_to_file(char*  fn_prefix_string,int id) {
print_weights_to_file_matrix(fn_prefix_string,id,"w3.txt", p_w3_value,
inputsizeinnumberofbits,HIDDEN_DIM); }
2668 void print_w4_to_file(char*  fn_prefix_string,int id) {
print_weights_to_file_matrix(fn_prefix_string,id,"w4.txt", p_w4_value,
inputsizeinnumberofbits,HIDDEN_DIM); }
2669 void print_b1_to_file(char*  fn_prefix_string,int id) {
print_weights_to_file_vector(fn_prefix_string,id,"b1.txt", p_b1_value,
HIDDEN_DIM); }
2670 void print_b2_to_file(char*  fn_prefix_string,int id) {
print_weights_to_file_vector(fn_prefix_string,id,"b2.txt", p_b2_value,
HIDDEN_DIM); }
2671 void print_b3_to_file(char*  fn_prefix_string,int id) {
print_weights_to_file_vector(fn_prefix_string,id,"b3.txt", p_b3_value,
HIDDEN_DIM); }
2672 void print_b4_to_file(char*  fn_prefix_string,int id) {
print_weights_to_file_vector(fn_prefix_string,id,"b4.txt", p_b4_value,
HIDDEN_DIM); }
2673 void print_w5_to_file(char*  fn_prefix_string,int id) {
print_weights_to_file_vector(fn_prefix_string,id,"w5.txt", p_w5_value,
HIDDEN_DIM); }
2674 void print_w6_to_file(char*  fn_prefix_string,int id) {
print_weights_to_file_vector(fn_prefix_string,id,"w6.txt", p_w6_value,
HIDDEN_DIM); }
2675 void print_b56_to_file(char* fn_prefix_string,int id) {
print_weights_to_file_single_number(fn_prefix_string,id,"b56.txt",p_b56_value
); }
2676 void print_loss_to_file(char* fn_prefix_string,int id) {
print_weights_to_file_single_number(fn_prefix_string,id,"loss.txt",p_loss_val
ue); }
2677
2678 void get_w0_from_file(char*  fn_prefix_string,int id) {
read_weights_to_file_vector(fn_prefix_string,id,"w0.txt", p_w0_value,
inputsizeinnumberofbits); }
2679 void get_w1_from_file(char*  fn_prefix_string,int id) {
read_weights_to_file_matrix(fn_prefix_string,id,"w1.txt", p_w1_value,
inputsizeinnumberofbits,HIDDEN_DIM); }
2680 void get_w2_from_file(char*  fn_prefix_string,int id) {
read_weights_to_file_matrix(fn_prefix_string,id,"w2.txt", p_w2_value,
inputsizeinnumberofbits,HIDDEN_DIM); }
```

```
2681 void get_w3_from_file(char*  fn_prefix_string,int id) {
read_weights_to_file_matrix(fn_prefix_string,id,"w3.txt", p_w3_value,
inputsizeinnumberofbits,HIDDEN_DIM); }
2682 void get_w4_from_file(char*  fn_prefix_string,int id) {
read_weights_to_file_matrix(fn_prefix_string,id,"w4.txt", p_w4_value,
inputsizeinnumberofbits,HIDDEN_DIM); }
2683 void get_b1_from_file(char*  fn_prefix_string,int id) {
read_weights_to_file_vector(fn_prefix_string,id,"b1.txt", p_b1_value,
HIDDEN_DIM); }
2684 void get_b2_from_file(char*  fn_prefix_string,int id) {
read_weights_to_file_vector(fn_prefix_string,id,"b2.txt", p_b2_value,
HIDDEN_DIM); }
2685 void get_b3_from_file(char*  fn_prefix_string,int id) {
read_weights_to_file_vector(fn_prefix_string,id,"b3.txt", p_b3_value,
HIDDEN_DIM); }
2686 void get_b4_from_file(char*  fn_prefix_string,int id) {
read_weights_to_file_vector(fn_prefix_string,id,"b4.txt", p_b4_value,
HIDDEN_DIM); }
2687 void get_w5_from_file(char*  fn_prefix_string,int id) {
read_weights_to_file_vector(fn_prefix_string,id,"w5.txt", p_w5_value,
HIDDEN_DIM); }
2688 void get_w6_from_file(char*  fn_prefix_string,int id) {
read_weights_to_file_vector(fn_prefix_string,id,"w6.txt", p_w6_value,
HIDDEN_DIM); }
2689 void get_b56_from_file(char* fn_prefix_string,int id) {
read_weights_to_file_single_number(fn_prefix_string,id,"b56.txt",p_b56_value)
; }
2690 void get_loss_from_file(char* fn_prefix_string,int id) {
read_weights_to_file_single_number(fn_prefix_string,id,"loss.txt",p_loss_valu
e); }
2691
2692 void print_all_weights_to_file(char*
fn_prefix_string_with_two_parameters,int id) {
2693   print_w0_to_file( fn_prefix_string_with_two_parameters,id);
2694   print_w1_to_file( fn_prefix_string_with_two_parameters,id);
2695   print_w2_to_file( fn_prefix_string_with_two_parameters,id);
2696   print_w3_to_file( fn_prefix_string_with_two_parameters,id);
2697   print_w4_to_file( fn_prefix_string_with_two_parameters,id);
2698   print_b1_to_file( fn_prefix_string_with_two_parameters,id);
2699   print_b2_to_file( fn_prefix_string_with_two_parameters,id);
2700   print_b3_to_file( fn_prefix_string_with_two_parameters,id);
2701   print_b4_to_file( fn_prefix_string_with_two_parameters,id);
2702   print_w5_to_file( fn_prefix_string_with_two_parameters,id);
2703   print_w6_to_file( fn_prefix_string_with_two_parameters,id);
2704   print_b56_to_file(fn_prefix_string_with_two_parameters,id);
2705   // print_loss_to_file(fn_prefix_string_with_two_parameters,id); // the
code for geting loss does not work; therefore, we don't call this one here.
2706 }
2707
2708 void get_all_weights_from_file(char*
fn_prefix_string_with_two_parameters,int id) {
2709   get_w0_from_file(fn_prefix_string_with_two_parameters,id);
2710   get_w1_from_file(fn_prefix_string_with_two_parameters,id);
2711   get_w2_from_file(fn_prefix_string_with_two_parameters,id);
2712   get_w3_from_file(fn_prefix_string_with_two_parameters,id);
2713   get_w4_from_file(fn_prefix_string_with_two_parameters,id);
2714   get_b1_from_file(fn_prefix_string_with_two_parameters,id);
```

```
2715   get_b2_from_file(fn_prefix_string_with_two_parameters,id);
2716   get_b3_from_file(fn_prefix_string_with_two_parameters,id);
2717   get_b4_from_file(fn_prefix_string_with_two_parameters,id);
2718   get_w5_from_file(fn_prefix_string_with_two_parameters,id);
2719   get_w6_from_file(fn_prefix_string_with_two_parameters,id);
2720   get_b56_from_file(fn_prefix_string_with_two_parameters,id);
2721   // get_loss_from_file(fn_prefix_string_with_two_parameters,id); // the
code for geting loss does not work; therefore, we don't call this one here.
2722 }
2723
2724 void change_dat_extension_to_txt_extention(char* fn) {
2725   int len;
2726   len = strlen(fn);
2727   if (fn[len-4]=='.') {
2728     if (fn[len-3]=='d') {
2729       if (fn[len-2]=='a') {
2730         if (fn[len-1]=='t') {
2731           fn[len-3] = 't';
2732           fn[len-2] = 'x';
2733           fn[len-1] = 't';
2734         } else { printf("Error in
change_dat_extension_to_txt_extention.\n"); exit(-1); }
2735       } else { printf("Error in
change_dat_extension_to_txt_extention.\n"); exit(-1); }
2736     } else { printf("Error in
change_dat_extension_to_txt_extention.\n"); exit(-1); }
2737   } else { printf("Error in change_dat_extension_to_txt_extention.\n");
exit(-1); }
2738 }
2739
2740 void do_fsync_on_file(char* fn) {
2741   int fd;
2742   fd = open(fn, O_RDONLY );
2743   if (fsync(fd)!=0) { printf("Error in do_fsync_on_file.\n"); exit(-1);
}
2744   close(fd);
2745 }
2746
2747 void do_fsync_on_current_directory() {
2748   int fd;
2749   fd = open(".", O_RDONLY );
2750   if (fsync(fd)!=0) { printf("Error in
do_fsync_on_current_directory.\n"); exit(-1); }
2751   close(fd);
2752 }
2753
2754 void write_input_to_file(char* outputfn,int* inputtoprogram) {
2755   int fd; int countwritten;
2756   fd = open(outputfn, O_RDWR | O_CREAT, S_IRUSR | S_IWUSR );
2757   if (fd==-1) { printf("Error when opening file\n"); exit(-1); }
2758   countwritten = write( fd, inputtoprogram,
getinputtoprogram_size_in_number_of_bytes() );
2759   if (countwritten==0) { printf("Error when writing to file.
write_input_to_file\n"); exit(-1); }
2760   close(fd);
2761   do_fsync_on_file(outputfn);
2762   do_fsync_on_current_directory();
```

```
2763 }
2764
2765 void write_input_to_txt_file(char* outputfn,int* inputtoprogram) {
2766   FILE* f; int i;
2767   f = fopen(outputfn,"w+");
2768   if (f==NULL) { printf("Error in write_input_to_txt_file.\n"); exit(-
1); }
2769   for (i=0;i<getinputtoprogram_size_in_number_of_ints();i++) {
2770     fprintf(f,"%d\n", inputtoprogram[i]);
2771   }
2772   fclose(f);
2773   do_fsync_on_file(outputfn);
2774   do_fsync_on_current_directory();
2775 }
2776
2777 void write_executiontime_to_file(char* outputfn,double executiontime) {
2778   FILE* f;
2779   f = fopen(outputfn, "w+");
2780   if (f==NULL) { printf("Error in write_executiontime_to_file.\n");
exit(-1); }
2781   fprintf(f,"%15.9lf", executiontime);
2782   fclose(f);
2783   do_fsync_on_file(outputfn);
2784   do_fsync_on_current_directory();
2785 }
2786
2787 void write_input_to_file_interpret_datfn_as_textfn(char*
outputfn_inputtoprogram,int* inputtoprogram) {
2788   char outputfn_inputtoprogram_text[200];
2789   strcpy(outputfn_inputtoprogram_text,outputfn_inputtoprogram);
2790   change_dat_extension_to_txt_extention(outputfn_inputtoprogram_text);
2791   write_input_to_txt_file(outputfn_inputtoprogram_text,inputtoprogram);
2792 }
2793
2794 void write_input_and_execution_time_to_files(char*
outputfn_inputtoprogram,char* outputfn_executiontime,int*
inputtoprogram,double executiontime) {
2795   write_input_to_file(
outputfn_inputtoprogram, inputtoprogram  );
2796   write_input_to_file_interpret_datfn_as_textfn(
outputfn_inputtoprogram, inputtoprogram  );
2797   write_executiontime_to_file(                    outputfn_executiontime,
executiontime   );
2798 }
2799
2800 void setprocessoraffinity_to_allow_just_a_single_processor_proc0() {
2801   cpu_set_t mask; int ret;
2802   CPU_ZERO(&mask);
2803   // CPU_SET(4,&mask); // we may want to disable hyperthreading
2804   CPU_SET(0,&mask); // we may want to disable hyperthreading
2805   ret = sched_setaffinity( 0, sizeof(mask), &mask);
2806   if (ret!=0) {
2807     printf("sched_setaffinity failed. ret = %d\n", ret);
2808     exit(-1);
2809   }
2810 }
2811
```

```
2812 void setprocessoraffinity_to_allow_all_processors() {
2813   cpu_set_t mask; int ret; int p;
2814   CPU_ZERO(&mask);
2815   // for (p=0;p<6;p++) {
2816   for (p=0;p<NUMBER_OF_PHYSICAL_PROCESSOR_CORES;p++) {
2817     CPU_SET(p,&mask); // we may want to disable hyperthreading
2818   }
2819   ret = sched_setaffinity( 0, sizeof(mask), &mask);
2820   if (ret!=0) {
2821     printf("sched_setaffinity failed. ret = %d\n", ret);
2822     exit(-1);
2823   }
2824 }
2825
2826 int get_bin_from_example(int i,int K) {
2827   double mint; double maxt; double averaget; double ratio; int
bin_index;
2828   get_min_max_average_execution_time(&mint,&maxt,&averaget);
2829   ratio = (manyruns[i].t-mint)/(maxt-mint);
2830   bin_index = K*ratio;
2831   if (bin_index>K-1) { // this will only happen for t=maxt
2832     bin_index=K-1;
2833   }
2834   return bin_index;
2835 }
2836
2837 void fill_frequency_of_each_bin_from_examples(int* p_n_examples,int*
frequency_of_each_bin,int K) {
2838   int i; int k;
2839   for (k=0;k<K;k++) { frequency_of_each_bin[k] = 0; }
2840   for (i=0;i<(*p_n_examples);i++) {
2841     k = get_bin_from_example(i,K);
2842     frequency_of_each_bin[k] = frequency_of_each_bin[k] + 1;
2843   }
2844 }
2845
2846 int find_smallest_frequency_among_bins(int* frequency_of_each_bin,int K)
{
2847   int smallest_frequency; int k;
2848   if (K<=0) { printf("Error in find_smallest_frequency_among_bins. K is
too small."); exit(-1); }
2849   smallest_frequency = frequency_of_each_bin[0];
2850   for (k=1;k<K;k++) { if (frequency_of_each_bin[k]<smallest_frequency) {
smallest_frequency = frequency_of_each_bin[k]; } }
2851   return smallest_frequency;
2852 }
2853
2854 void swap_example_inputtoprogram(int lo_index,int hi_index) {
2855   int temp; int tempindex;
2856   for
(tempindex=0;tempindex<getinputtoprogram_size_in_number_of_ints();tempindex++
) {
2857     temp = manyruns[lo_index].inputtoprogram[tempindex];
2858     manyruns[lo_index].inputtoprogram[tempindex] =
manyruns[hi_index].inputtoprogram[tempindex];
2859     manyruns[hi_index].inputtoprogram[tempindex] = temp;
2860   }
```

```
2861 }
2862 void swap_example_t(int lo_index,int hi_index) {
2863   double temp;
2864   temp = manyruns[lo_index].t;
2865   manyruns[lo_index].t = manyruns[hi_index].t;
2866   manyruns[hi_index].t = temp;
2867 }
2868 void swap_example(int lo_index,int hi_index) {
2869   swap_example_inputtoprogram(lo_index,hi_index);
2870   swap_example_t(lo_index,hi_index);
2871 }
2872
2873 void do_the_thinning_of_examples_with_given_K_and_smallest_frequency(int
K,int smallest_frequency,int* p_n_examples) {
2874   int i; int k; int* count_of_each_bin; int lo_index; int hi_index;
2875   count_of_each_bin = malloc(K*sizeof(int)); if
(count_of_each_bin==NULL) { printf("Memory allocation failure in
do_the_thinning_of_examples_with_given_K_and_smallest_frequency.\n"); exit(-
1); }
2876   for (k=0;k<K;k++) { count_of_each_bin[k] = 0; }
2877   for (i=0;i<(*p_n_examples);i++) {
2878     k = get_bin_from_example(i,K);
2879     if (count_of_each_bin[k]<smallest_frequency) {
2880       count_of_each_bin[k] = count_of_each_bin[k] + 1;
2881     } else {
2882       manyruns[i].t = -manyruns[i].t; // this is our way of marking that
we should not consider this training example
2883     }
2884   }
2885   free(count_of_each_bin);
2886
2887   lo_index = 0;
2888   hi_index = (*p_n_examples)-1;
2889   do {
2890     if (manyruns[lo_index].t>0.0) {
2891       if (manyruns[hi_index].t>0.0) {
2892         lo_index = lo_index + 1;
2893       } else {
2894         lo_index = lo_index + 1;
2895         hi_index = hi_index - 1;
2896       }
2897     } else {
2898       if (manyruns[hi_index].t>0.0) {
2899         swap_example(lo_index,hi_index);
2900         lo_index = lo_index + 1;
2901         hi_index = hi_index - 1;
2902       } else {
2903         hi_index = hi_index - 1;
2904       }
2905     }
2906   } while (lo_index<hi_index);
2907   if (lo_index==hi_index) {
2908     if (manyruns[lo_index].t>0.0) {
2909       lo_index = lo_index + 1;
2910     } else {
2911       hi_index = hi_index - 1;
2912     }
```

```
2913    }
2914    for (i=0;i<(*p_n_examples);i++) {
2915      if (manyruns[i].t<0.0) {
2916        manyruns[i].t = -manyruns[i].t; // restore
2917      }
2918    }
2919    (*p_n_examples) = lo_index;
2920  }
2921
2922  int get_smallest_frequency_assuming_no_frequency_table(int*
p_n_examples,int K) {
2923    int* frequency_of_each_bin; int smallest_frequency;
2924    frequency_of_each_bin = malloc(K*sizeof(int)); if
(frequency_of_each_bin==NULL) { printf("Memory allocation failure in
get_smallest_frequency_assuming_no_frequency_table.\n"); exit(-1); }
2925
fill_frequency_of_each_bin_from_examples(p_n_examples,frequency_of_each_bin,K
);
2926    smallest_frequency =
find_smallest_frequency_among_bins(frequency_of_each_bin,K);
2927    free(frequency_of_each_bin);
2928    return smallest_frequency;
2929  }
2930
2931  // find the largest K such that
target_at_least_n_elements_in_bucket<=smallest_frequency
2932  int find_K(int* p_n_examples,int target_at_least_n_elements_in_bucket) {
2933    int K; int smallest_frequency; int lb_K; int ub_K;
2934    K = 1;
2935    smallest_frequency =
get_smallest_frequency_assuming_no_frequency_table(p_n_examples,K);
2936    if (target_at_least_n_elements_in_bucket<=smallest_frequency) {
2937      lb_K = 1;
2938      ub_K = (*p_n_examples);
2939      while (lb_K<ub_K) {
2940        if (lb_K+1==ub_K) {
2941          K = lb_K+1;
2942          smallest_frequency =
get_smallest_frequency_assuming_no_frequency_table(p_n_examples,K);
2943          if (target_at_least_n_elements_in_bucket<=smallest_frequency) {
2944            lb_K = lb_K + 1;
2945          } else {
2946            ub_K = ub_K - 1;
2947          }
2948        } else {
2949          K = (lb_K+ub_K)/2;
2950          smallest_frequency =
get_smallest_frequency_assuming_no_frequency_table(p_n_examples,K);
2951          if (target_at_least_n_elements_in_bucket<=smallest_frequency) {
2952            lb_K = K;
2953          } else {
2954            ub_K = K;
2955          }
2956        }
2957      }
2958      return lb_K;
2959    } else {
```

```
2960     printf("Error in find_K. smallest_frequency is too small."); exit(-
1);
2961   }
2962 }
2963
2964 void do_the_thinning_of_examples_with_given_K(int K,int*
p_n_examples,int target_at_least_n_elements_in_bucket) {
2965   int smallest_frequency;
2966   smallest_frequency =
get_smallest_frequency_assuming_no_frequency_table(p_n_examples,K);
2967
do_the_thinning_of_examples_with_given_K_and_smallest_frequency(K,smallest_fr
equency,p_n_examples);
2968 }
2969
2970 // Consider the case that we have 30000000 examples.
2971 //   If we set K=4, then after thinning, we get n_examples = 1310540
2972 //   If we set K=5, then after thinning, we get n_examples =  343650
2973
2974 void thinning_examples(int* p_n_examples) {
2975   int K; int target_at_least_n_elements_in_bucket;
2976   // target_at_least_n_elements_in_bucket = 1000;
2977   // K = find_K(p_n_examples,target_at_least_n_elements_in_bucket);
2978   // K = 4;
2979   K = 5;
2980   printf("Found K=%d\n", K);
2981   if (K<1) {
2982     printf("Error in thinning_examples. K is too small."); exit(-1);
2983   }
2984
do_the_thinning_of_examples_with_given_K(K,p_n_examples,target_at_least_n_ele
ments_in_bucket);
2985 }
2986
2987 int cmpfunc_sort_index_of_examples(const void * a, const void * b) {
2988   int index_a; int index_b;
2989   double t_a; double t_b;
2990   index_a = *((int*)a);
2991   index_b = *((int*)b);
2992   t_a = manyruns[index_a].t;
2993   t_b = manyruns[index_b].t;
2994   if (t_a < t_b) {
2995     return -1;
2996   } else {
2997     if (t_a == t_b) {
2998       return 0;
2999     } else {
3000       return 1;
3001     }
3002   }
3003 }
3004
3005 void fill_index_of_examples(int n_examples) {
3006   int i;
3007   for (i=0;i<n_examples;i++) {
3008     index_of_examples[i] = i;
3009   }
```

```
3010   qsort(index_of_examples, n_examples, sizeof(int),
cmpfunc_sort_index_of_examples);
3011 }
3012
3013 TF_Buffer* ReadFile(const char* filename) {
3014   int fd = open(filename, 0);
3015   if (fd < 0) {
3016     perror("failed to open file: ");
3017     return NULL;
3018   }
3019   struct stat stat;
3020   if (fstat(fd, &stat) != 0) {
3021     perror("failed to read file: ");
3022     return NULL;
3023   }
3024   char* data = (char*)malloc(stat.st_size);
3025   ssize_t nread = read(fd, data, stat.st_size);
3026   if (nread < 0) {
3027     perror("failed to read file: ");
3028     free(data);
3029     return NULL;
3030   }
3031   if (nread != stat.st_size) {
3032     fprintf(stderr, "read %zd bytes, expected to read %zd\n", nread,
3033             stat.st_size);
3034     free(data);
3035     return NULL;
3036   }
3037   TF_Buffer* ret = TF_NewBufferFromString(data, stat.st_size);
3038   free(data);
3039   return ret;
3040 }
3041
3042 void print_all_operations_of_graph(TF_Graph* g) {
3043   size_t pos = 0;
3044   TF_Operation* oper;
3045   while ((oper = TF_GraphNextOperation(g, &pos)) != NULL) {
3046     printf("%s\n",TF_OperationName(oper));
3047   }
3048 }
3049
3050 int ModelCreate(model_t* model, const char* graph_def_filename) {
3051   model->status = TF_NewStatus();
3052   model->graph = TF_NewGraph();
3053
3054   {
3055     // Create the session.
3056     TF_SessionOptions* opts = TF_NewSessionOptions();
3057     model->session = TF_NewSession(model->graph, opts, model->status);
3058     TF_DeleteSessionOptions(opts);
3059     if (!Okay(model->status)) return 0;
3060   }
3061
3062   TF_Graph* g = model->graph;
3063
3064   {
3065     // Import the graph.
```

```
3066      TF_Buffer* graph_def = ReadFile(graph_def_filename);
3067      if (graph_def == NULL) return 0;
3068      printf("Read GraphDef of %zu bytes\n", graph_def->length);
3069      TF_ImportGraphDefOptions* opts = TF_NewImportGraphDefOptions();
3070      TF_GraphImportGraphDef(g, graph_def, opts, model->status);
3071      TF_DeleteImportGraphDefOptions(opts);
3072      TF_DeleteBuffer(graph_def);
3073      if (!Okay(model->status)) return 0;
3074    }
3075
3076    // Handles to the interesting operations in the graph.
3077    model->input.oper = TF_GraphOperationByName(g, "input");
3078    model->input.index = 0;
3079    model->target.oper = TF_GraphOperationByName(g, "target");
3080    model->target.index = 0;
3081    model->output.oper = TF_GraphOperationByName(g, "output");
3082    model->output.index = 0;
3083
3084    model->init_op = TF_GraphOperationByName(g, "init");
3085    model->train_op = TF_GraphOperationByName(g, "train");
3086    model->save_op = TF_GraphOperationByName(g,
"save/control_dependency");
3087    model->restore_op = TF_GraphOperationByName(g, "save/restore_all");
3088
3089    model->checkpoint_file.oper = TF_GraphOperationByName(g,
"save/Const");
3090    model->checkpoint_file.index = 0;
3091
3092    return 1;
3093  }
3094
3095  int ModelInit(model_t* model) {
3096    const TF_Operation* init_op[1] = {model->init_op};
3097    TF_SessionRun(model->session, NULL,
3098                  /* No inputs */
3099                  NULL, NULL, 0,
3100                  /* No outputs */
3101                  NULL, NULL, 0,
3102                  /* Just the init operation */
3103                  init_op, 1,
3104                  /* No metadata */
3105                  NULL, model->status);
3106    return Okay(model->status);
3107  }
3108
3109  void ModelDestroy(model_t* model) {
3110    TF_DeleteSession(model->session, model->status);
3111    Okay(model->status);
3112    TF_DeleteGraph(model->graph);
3113    TF_DeleteStatus(model->status);
3114  }
3115
3116  int ModelPredictService(model_t* model, double* batch, int
batch_size,double* predictions,size_t nbytes_out) {
3117    const int64_t dims_in[2] = {batch_size, inputsizeinnumberofbits};
3118    const size_t nbytes_in = batch_size * sizeof(double) *
inputsizeinnumberofbits;
```

```
3119
3120    TF_Tensor* t_in = TF_AllocateTensor(TF_DOUBLE, dims_in, 2, nbytes_in);
3121
3122    memcpy(TF_TensorData(t_in), batch, nbytes_in);
3123
3124    TF_Output inputs[1] = {model->input};
3125    TF_Tensor* input_values[1] = {t_in};
3126    TF_Output outputs[1] = {model->output};
3127    TF_Tensor* output_values[1] = {NULL};
3128
3129    TF_SessionRun(model->session, NULL, inputs, input_values, 1, outputs,
3130                  output_values, 1,
3131                  NULL, 0, NULL, model->status);
3132
3133    TF_DeleteTensor(t_in);
3134    if (!Okay(model->status)) return 0;
3135
3136    if (TF_TensorByteSize(output_values[0]) != nbytes_out) {
3137      fprintf(stderr,
3138              "ERROR: Expected predictions tensor to have %zu bytes, has
%zu\n",
3139              nbytes_out, TF_TensorByteSize(output_values[0]));
3140      TF_DeleteTensor(output_values[0]);
3141      return 0;
3142    }
3143    memcpy(predictions, TF_TensorData(output_values[0]), nbytes_out);
3144    TF_DeleteTensor(output_values[0]);
3145    return 1;
3146 }
3147
3148 #define BATCH_SIZE_FOR_PREDICT_ALL_EXAMPLES 1024
3149 int ModelPredictAllExamples(model_t* model,int n_examples,char*
fn_prefix_string,int id) {
3150    int startindex;
3151    int endindex;
3152    int batch_size;
3153    FILE* f_preds;
3154    FILE* f_errors;
3155    double* batch;
3156    size_t nbytes_out;
3157    double* predictions;
3158    double* errors;
3159    int i; int j;
3160    char fn_to_use[2000];
3161    sprintf(fn_to_use,"%s_%d_%s",fn_prefix_string,id,"preds");
3162    f_preds     = fopen(fn_to_use, "wb" );
if (f_preds==NULL)           { printf("Error in ModelPredictAllExamples.
preds\n");               exit(-1); }
3163    sprintf(fn_to_use,"%s_%d_%s",fn_prefix_string,id,"errors");
3164    f_errors    = fopen(fn_to_use, "wb" );
if (f_errors==NULL)          { printf("Error in ModelPredictAllExamples.
errors\n");              exit(-1); }
3165    batch       =
malloc(BATCH_SIZE_FOR_PREDICT_ALL_EXAMPLES*inputsizeinnumberofbits*sizeof(dou
ble)); if (batch==NULL)             { printf("Memory allocation failure in
ModelPredictAllExamples.\n"); exit(-1); }
```

```
3166   predictions = malloc(BATCH_SIZE_FOR_PREDICT_ALL_EXAMPLES *
sizeof(double));                        if (predictions==NULL)       {
printf("Memory allocation failure in ModelPredictAllExamples.\n"); exit(-1);
}
3167   errors      = malloc(BATCH_SIZE_FOR_PREDICT_ALL_EXAMPLES *
sizeof(double));                        if (errors==NULL)            {
printf("Memory allocation failure in ModelPredictAllExamples.\n"); exit(-1);
}
3168   startindex = 0;
3169   while (startindex<n_examples) {
3170     endindex = startindex + BATCH_SIZE_FOR_PREDICT_ALL_EXAMPLES-1;
3171     if (endindex>n_examples-1) { endindex=n_examples-1;}
3172     batch_size = endindex - startindex + 1;
3173     for (i=0;i<batch_size;i++) {
3174       for (j=0;j<inputsizeinnumberofbits;j++) {
3175         if
(isbitonein_bitposition_in_integerarray(manyruns[startindex+i].inputtoprogram
,j)) {
3176           batch[i*inputsizeinnumberofbits+j] = 1.0;
3177         } else {
3178           batch[i*inputsizeinnumberofbits+j] = 0.0;
3179         }
3180       }
3181     }
3182     nbytes_out = batch_size * sizeof(double);
3183     if
(!ModelPredictService(model,batch,batch_size,predictions,nbytes_out)) {
free(errors);free(predictions);free(batch);fclose( f_preds);fclose(
f_errors); return 1; }
3184     for (i=0;i<batch_size;i++) {
3185       errors[i] = predictions[i]-manyruns[startindex+i].t;
3186     }
3187     fwrite(predictions,sizeof(double),batch_size,f_preds);
3188     fwrite(errors,     sizeof(double),batch_size,f_errors);
3189     startindex = startindex + BATCH_SIZE_FOR_PREDICT_ALL_EXAMPLES;
3190   }
3191   free(errors);
3192   free(predictions);
3193   free(batch);
3194   fclose( f_preds);
3195   fclose( f_errors);
3196   return 1;
3197 }
3198
3199 int ModelPredictAllExamples_specialized_for_uniform_time(model_t*
model,int n_examples,char* fn_prefix_string,int id) {
3200   int startindex;
3201   int endindex;
3202   int batch_size;
3203   double* batch;
3204   size_t nbytes_out;
3205   double* predictions;
3206   double* errors;
3207   int i; int j;
3208   char fn_to_use[2000];
3209   int row_in_dataset;
3210   int* rows_in_dataset;
```

```
3211   double sum; int count;
3212   sum = 0.0; count = 0;
3213   rows_in_dataset =
malloc(BATCH_SIZE_FOR_PREDICT_ALL_EXAMPLES*sizeof(int));
if (rows_in_dataset==NULL)   { printf("Memory allocation failure in
ModelPredictAllExamples_uniform_time.\n"); exit(-1); }
3214   batch       =
malloc(BATCH_SIZE_FOR_PREDICT_ALL_EXAMPLES*inputsizeinnumberofbits*sizeof(dou
ble)); if (batch==NULL)             { printf("Memory allocation failure in
ModelPredictAllExamples_uniform_time.\n"); exit(-1); }
3215   predictions = malloc(BATCH_SIZE_FOR_PREDICT_ALL_EXAMPLES *
sizeof(double));                    if (predictions==NULL)       {
printf("Memory allocation failure in
ModelPredictAllExamples_uniform_time.\n"); exit(-1); }
3216   errors      = malloc(BATCH_SIZE_FOR_PREDICT_ALL_EXAMPLES *
sizeof(double));                    if (errors==NULL)            {
printf("Memory allocation failure in
ModelPredictAllExamples_uniform_time.\n"); exit(-1); }
3217   startindex = 0;
3218   while (startindex<n_examples) {
3219     endindex = startindex + BATCH_SIZE_FOR_PREDICT_ALL_EXAMPLES-1;
3220     if (endindex>n_examples-1) { endindex=n_examples-1;}
3221     batch_size = endindex - startindex + 1;
3222     for (i=0;i<batch_size;i++) {
3223       rows_in_dataset[i] =
obtain_row_using_uniform_sampling_over_execution_time(n_examples);
3224     }
3225     for (i=0;i<batch_size;i++) {
3226       for (j=0;j<inputsizeinnumberofbits;j++) {
3227         if
(isbitonein_bitposition_in_integerarray(manyruns[rows_in_dataset[i]].inputtop
rogram,j)) {
3228           batch[i*inputsizeinnumberofbits+j] = 1.0;
3229         } else {
3230           batch[i*inputsizeinnumberofbits+j] = 0.0;
3231         }
3232       }
3233     }
3234     nbytes_out = batch_size * sizeof(double);
3235     if
(!ModelPredictService(model,batch,batch_size,predictions,nbytes_out)) {
free(errors);free(predictions);free(batch);free(rows_in_dataset); return 1; }
3236     for (i=0;i<batch_size;i++) {
3237       errors[i] = predictions[i]-manyruns[rows_in_dataset[i]].t;
3238     }
3239     for (i=0;i<batch_size;i++) {
3240       sum = sum + fabs(errors[i]); count = count + 1;
3241     }
3242     startindex = startindex + BATCH_SIZE_FOR_PREDICT_ALL_EXAMPLES;
3243   }
3244   neural_network_average_abs_error_uniform_time = sum/count;
3245   free(errors);
3246   free(predictions);
3247   free(batch);
3248   free(rows_in_dataset);
3249   return 1;
3250 }
```

```
3251 #undef BATCH_SIZE_FOR_PREDICT_ALL_EXAMPLES
3252
3253 void updatemin(double* p_min_value,int* p_min_value_index,double
in_value,int i) {
3254   if ((*p_min_value)>in_value) {
3255     (*p_min_value)=in_value;
3256     (*p_min_value_index)=i;
3257   }
3258 }
3259 void updatemax(double* p_max_value,int* p_max_value_index,double
in_value,int i) {
3260   if ((*p_max_value)<in_value) {
3261     (*p_max_value)=in_value;
3262     (*p_max_value_index)=i;
3263   }
3264 }
3265 void calculate_statistics_on_predictions_and_prediction_errors(int
n_examples,char* fn_prefix_string,int id) {
3266   int i;
3267   FILE* f_preds;
3268   FILE* f_errors;
3269   double* predictions;
3270   double* errors;
3271   double min_prediction; int index_of_min_prediction;
3272   double max_prediction; int index_of_max_prediction;
3273   double min_error; int index_of_min_error;
3274   double max_error; int index_of_max_error;
3275   double max_abs_error; int index_of_max_abs_error;
3276   double sum_abs_error;
3277   char fn_to_use[2000];
3278   int n_read;
3279   FILE* f_pred_and_error_statistics;
3280   sprintf(fn_to_use,"%s_%d_%s",fn_prefix_string,id,"preds");
3281   f_preds     = fopen(fn_to_use, "rb" );             if (f_preds==NULL)
{ printf("Error in ModelPredictAllExamples. preds\n");               exit(-
1); }
3282   sprintf(fn_to_use,"%s_%d_%s",fn_prefix_string,id,"errors");
3283   f_errors    = fopen(fn_to_use, "rb" );             if (f_errors==NULL)
{ printf("Error in ModelPredictAllExamples. errors\n");              exit(-
1); }
3284   predictions = malloc(n_examples * sizeof(double)); if
(predictions==NULL)       { printf("Memory allocation failure in
ModelPredictAllExamples.\n"); exit(-1); }
3285   errors      = malloc(n_examples * sizeof(double)); if (errors==NULL)
{ printf("Memory allocation failure in ModelPredictAllExamples.\n"); exit(-
1); }
3286   n_read=fread(predictions,sizeof(double),n_examples,f_preds);  if
(n_read!=n_examples) { printf("Read failure in ModelPredictAllExamples.
pred\n"); exit(-1); }
3287   n_read=fread(errors,     sizeof(double),n_examples,f_errors); if
(n_read!=n_examples) { printf("Read failure in ModelPredictAllExamples.
errors\n"); exit(-1); }
3288   min_prediction=predictions[0]; index_of_min_prediction=0;
3289   max_prediction=predictions[0]; index_of_max_prediction=0;
3290   min_error     =errors[0];      index_of_min_error     =0;
3291   max_error     =errors[0];      index_of_max_error     =0;
3292   max_abs_error =fabs(errors[0]);index_of_max_abs_error =0;
```

```
3293   sum_abs_error =0.0;
3294   for (i=1;i<n_examples;i++) {
3295     updatemin(&min_prediction,&index_of_min_prediction,predictions[i],
i);
3296     updatemax(&max_prediction,&index_of_max_prediction,predictions[i],
i);
3297     updatemin(&min_error,     &index_of_min_error,     errors[i],
i);
3298     updatemax(&max_error,     &index_of_max_error,     errors[i],
i);
3299     updatemax(&max_abs_error, &index_of_max_abs_error,
fabs(errors[i]),i);
3300     sum_abs_error  = sum_abs_error + fabs(errors[i]);
3301   }
3302
sprintf(fn_to_use,"%s_%d_%s",fn_prefix_string,id,"pred_and_error_statistics.t
xt");
3303   f_pred_and_error_statistics = fopen(fn_to_use, "w" ); if
(f_pred_and_error_statistics==NULL) { printf("Error in
ModelPredictAllExamples. f_pred_and_error_statistics\n"); exit(-1); }
3304
fprintf(f_pred_and_error_statistics,"min_prediction=%21.18lf\n",min_predictio
n);
fprintf(f_pred_and_error_statistics,"index_of_min_prediction=%d\n",index_of_m
in_prediction);
3305
fprintf(f_pred_and_error_statistics,"max_prediction=%21.18lf\n",max_predictio
n);
fprintf(f_pred_and_error_statistics,"index_of_max_prediction=%d\n",index_of_m
ax_prediction);
3306   fprintf(f_pred_and_error_statistics,"min_error=%21.18lf\n",min_error);
fprintf(f_pred_and_error_statistics,"index_of_min_error=%d\n",index_of_min_er
ror);
3307   fprintf(f_pred_and_error_statistics,"max_error=%21.18lf\n",max_error);
fprintf(f_pred_and_error_statistics,"index_of_max_error=%d\n",index_of_max_er
ror);
3308
fprintf(f_pred_and_error_statistics,"max_abs_error=%21.18lf\n",max_abs_error)
;
fprintf(f_pred_and_error_statistics,"index_of_max_abs_error=%d\n",index_of_ma
x_abs_error);
3309   fprintf(f_pred_and_error_statistics,"avg_abs_error=
%21.18lf\n",sum_abs_error/n_examples);
3310
fprintf(f_pred_and_error_statistics,"neural_network_average_abs_error_uniform
_time= %21.18lf\n",neural_network_average_abs_error_uniform_time);
3311   fclose(f_pred_and_error_statistics);
3312   free(errors);
3313   free(predictions);
3314   fclose( f_preds);
3315   fclose( f_errors);
3316 }
3317
3318 int ModelPredict(model_t* model, double* batch, int batch_size) {
3319   int i;
3320   const size_t nbytes_out = batch_size * sizeof(double);
```

```
3321   double* predictions = malloc(nbytes_out); if (predictions==NULL) {
printf("Memory allocation failure in ModelPredict."); exit(-1); }
3322   if
(!ModelPredictService(model,batch,batch_size,predictions,nbytes_out)) {
free(predictions); return 1; }
3323   printf("Predictions:\n");
3324   for (i=0;i<batch_size;i++) {
3325     printf("\t first_input = %lf, second_input = %lf, third_input = %lf,
predicted y = %lf\n", batch[inputsizeinnumberofbits*i],
batch[inputsizeinnumberofbits*i+1], batch[inputsizeinnumberofbits*i+2],
predictions[i]);
3326   }
3327   free(predictions);
3328   return 1;
3329 }
3330
3331 void fillobj_for_objective_function() {
3332   int j;
3333   for (j=0;j<inputsizeinnumberofbits;j++) {
3334     obj[j]   = 0.0;
3335     lb[j]    = 0.0;
3336     ub[j]    = 1.0;
3337     vtype[j] = GRB_BINARY;
3338   }
3339   for
(j=inputsizeinnumberofbits;j<inputsizeinnumberofbits+HIDDEN_DIM*6;j++) {
3340     obj[j]   = 0.0;
3341     lb[j]    = -GRB_INFINITY;
3342     ub[j]    = GRB_INFINITY;
3343     vtype[j] = GRB_CONTINUOUS;
3344   }
3345   j = inputsizeinnumberofbits+HIDDEN_DIM*6;
3346   obj[j]   = 1.0;
3347   lb[j]    = -GRB_INFINITY;
3348   ub[j]    = GRB_INFINITY;
3349   vtype[j] = GRB_CONTINUOUS;
3350 }
3351
3352 void setup_constraints_define_zterm_service_for_terms(int k,int
index,double* p_w_value,double* p_b_value) {
3353   int j;
3354   nelements_of_constr = 0;
3355   for (j=0;j<inputsizeinnumberofbits;j++) {
3356     ind[nelements_of_constr] = j;
3357     val[nelements_of_constr] = 2*p_w_value[j*HIDDEN_DIM+k];
3358     nelements_of_constr = nelements_of_constr + 1;
3359   }
3360   j = inputsizeinnumberofbits+index*HIDDEN_DIM+k;
3361   ind[nelements_of_constr] = j;
3362   val[nelements_of_constr] = -1.0;
3363   nelements_of_constr = nelements_of_constr+1;
3364   // rhs = -p_b1_value[k];
3365   rhs = -p_b_value[k];
3366   for (j=0;j<inputsizeinnumberofbits;j++) {
3367     rhs = rhs+p_w_value[j*HIDDEN_DIM+k];
3368   }
3369 }
```

```
3370
3371 void setup_constraints_define_zmin_first_term(int k) {
3372
setup_constraints_define_zterm_service_for_terms(k,0,p_w1_value,p_b1_value);
3373 }
3374 void setup_constraints_define_zmin_second_term(int k) {
3375
setup_constraints_define_zterm_service_for_terms(k,1,p_w2_value,p_b2_value);
3376 }
3377
3378 void setup_constraints_define_zmax_first_term(int k) {
3379
setup_constraints_define_zterm_service_for_terms(k,2,p_w3_value,p_b3_value);
3380 }
3381 void setup_constraints_define_zmax_second_term(int k) {
3382
setup_constraints_define_zterm_service_for_terms(k,3,p_w4_value,p_b4_value);
3383 }
3384 double calc_UB_of_zmax_neuron(int k) {
3385   int j; double sum;
3386   sum = 0.0;
3387   for (j=0;j<inputsizeinnumberofbits;j++) {
3388     sum = sum + fabs(p_w3_value[j*HIDDEN_DIM+k]);
3389   }
3390   sum = sum + fabs(p_b3_value[k]);
3391   for (j=0;j<inputsizeinnumberofbits;j++) {
3392     sum = sum + fabs(p_w4_value[j*HIDDEN_DIM+k]);
3393   }
3394   sum = sum + fabs(p_b4_value[k]);
3395   return sum;
3396 }
3397
3398 void setup_constraints_define_y() {
3399   int j; int k;
3400   nelements_of_constr = 0;
3401   for (j=0;j<inputsizeinnumberofbits;j++) {
3402     ind[nelements_of_constr] = j;
3403     val[nelements_of_constr] = 2*p_w0_value[j];
3404     nelements_of_constr = nelements_of_constr + 1;
3405   }
3406   for (k=0;k<HIDDEN_DIM;k++) {
3407     j = inputsizeinnumberofbits+4*HIDDEN_DIM+k;
3408     ind[nelements_of_constr] = j;
3409     val[nelements_of_constr] = p_w5_value[k];
3410     nelements_of_constr = nelements_of_constr+1;
3411   }
3412   for (k=0;k<HIDDEN_DIM;k++) {
3413     j = inputsizeinnumberofbits+5*HIDDEN_DIM+k;
3414     ind[nelements_of_constr] = j;
3415     val[nelements_of_constr] = p_w6_value[k];
3416     nelements_of_constr = nelements_of_constr+1;
3417   }
3418   j = inputsizeinnumberofbits+6*HIDDEN_DIM;
3419   ind[nelements_of_constr] = j;
3420   val[nelements_of_constr] = -1.0;
3421   nelements_of_constr = nelements_of_constr+1;
3422
```

```
3423    rhs = -(*p_b56_value);
3424    for (j=0;j<inputsizeinnumberofbits;j++) {
3425      rhs = rhs+p_w0_value[j];
3426    }
3427  }
3428
3429  void find_WCET_input_from_prediction_model(int n_examples,char*
fn_prefix_string,int* an_int_array,double* p_t) {
3430    int j; int k; int index1; int index2; int selected_index;
3431    char gurobi_wcet_log_file[200];
3432    char gurobi_wcet_lp_file[200];
3433    char gurobi_wcet_sol_file[200];
3434    sprintf(gurobi_wcet_log_file,"%s_wcet.log",fn_prefix_string);
3435    sprintf(gurobi_wcet_lp_file, "%s_wcet.lp", fn_prefix_string);
3436    sprintf(gurobi_wcet_sol_file,"%s_wcet.sol",fn_prefix_string);
3437    nelements_of_obj = inputsizeinnumberofbits+HIDDEN_DIM*6+1;
3438    dothememoryallocation_for_Gurobi_stuff();
3439    error = GRBloadenv(&env, gurobi_wcet_log_file);
3440    if (error) goto QUIT;
3441    error = GRBnewmodel(env, &model, "wcet", 0, NULL, NULL, NULL, NULL,
NULL);
3442    if (error) goto QUIT;
3443    error = GRBsetdblparam(GRBgetenv(model), "IntFeasTol", 0.000000001);
3444    if (error) goto QUIT;
3445    error = GRBsetdblparam(GRBgetenv(model), GRB_DBL_PAR_TIMELIMIT,
5.00*3600.0); // 5 hours
3446    if (error) goto QUIT;
3447    fillobj_for_objective_function();
3448    error = GRBaddvars( model, nelements_of_obj, 0, NULL, NULL, NULL, obj,
lb, ub, vtype, NULL);
3449    if (error) goto QUIT;
3450    error = GRBsetintattr( model, GRB_INT_ATTR_MODELSENSE, GRB_MAXIMIZE);
3451    if (error) goto QUIT;
3452    error = GRBupdatemodel( model);
3453    if (error) goto QUIT;
3454    for (k=0;k<HIDDEN_DIM;k++) {
3455      setup_constraints_define_zmin_first_term(k);
3456      error = GRBaddconstr( model, nelements_of_constr, ind, val,
GRB_EQUAL, rhs, NULL );
3457    }
3458    for (k=0;k<HIDDEN_DIM;k++) {
3459      setup_constraints_define_zmin_second_term(k);
3460      error = GRBaddconstr( model, nelements_of_constr, ind, val,
GRB_EQUAL, rhs, NULL );
3461    }
3462    for (k=0;k<HIDDEN_DIM;k++) {
3463      ind[0] = inputsizeinnumberofbits+0*HIDDEN_DIM+k;
3464      ind[1] = inputsizeinnumberofbits+1*HIDDEN_DIM+k;
3465      error =
GRBaddgenconstrMin(model,NULL,inputsizeinnumberofbits+4*HIDDEN_DIM+k,2,ind,
GRB_INFINITY );
3466    }
3467    for (k=0;k<HIDDEN_DIM;k++) {
3468      setup_constraints_define_zmax_first_term(k);
3469      error = GRBaddconstr( model, nelements_of_constr, ind, val,
GRB_EQUAL, rhs, NULL );
3470    }
```

```
3471    for (k=0;k<HIDDEN_DIM;k++) {
3472      setup_constraints_define_zmax_second_term(k);
3473      error = GRBaddconstr( model, nelements_of_constr, ind, val,
GRB_EQUAL, rhs, NULL );
3474    }
3475    for (k=0;k<HIDDEN_DIM;k++) {
3476      ind[0] = inputsizeinnumberofbits+2*HIDDEN_DIM+k;
3477      ind[1] = inputsizeinnumberofbits+3*HIDDEN_DIM+k;
3478      error =
GRBaddgenconstrMax(model,NULL,inputsizeinnumberofbits+5*HIDDEN_DIM+k,2,ind, -
GRB_INFINITY );
3479    }
3480    setup_constraints_define_y();
3481    error = GRBaddconstr( model, nelements_of_constr, ind, val, GRB_EQUAL,
rhs, NULL );
3482    error = GRBupdatemodel( model);
3483    if (error) goto QUIT;
3484    error = GRBwrite( model,gurobi_wcet_lp_file);
3485    if (error) goto QUIT;
3486    error = GRBoptimize(model);
3487    if (error) goto QUIT;
3488    error = GRBupdatemodel( model);
3489    if (error) goto QUIT;
3490    error = GRBwrite( model,gurobi_wcet_sol_file);
3491    if (error) goto QUIT;
3492    error = GRBgetintattr( model, GRB_INT_ATTR_STATUS, &optimstatus);
3493    if (error) goto QUIT;
3494    if ((optimstatus == GRB_OPTIMAL) || (optimstatus == GRB_TIME_LIMIT)) {
3495      printf("optimstatus is optimal or timelimit\n"); fflush(stdout);
3496      error = GRBgetdblattr( model, GRB_DBL_ATTR_OBJVAL, &objval);
3497      if (error) goto QUIT;
3498      error = GRBgetdblattrarray( model, GRB_DBL_ATTR_X, 0,
nelements_of_obj, sol);
3499      if (error) goto QUIT;
3500      clear_int_array(an_int_array);
3501      for (j=0;j<inputsizeinnumberofbits;j++) {
3502        if (sol[j]>0.999) { // ideally, sol[j] is either 0 or 1 but
because of rounding effects, it can happen that sol[j] takes some other
values slightly different from 0 or 1.
3503          setbitonein_bitposition_in_integerarray(an_int_array,j);
3504        } else {
3505          if (sol[j]<0.001) {
3506            ; // this should be fine. We don't need to do anything because
we initialized the array with zeros
3507          } else {
3508            // printf("An error in
find_input_that_maximizes_predicted_execution_time_allow_twobitbasisfunctions
. A solution was obtained with a binary variable that was not 0 and not
1.\n"); fflush(stdout); exit(-1);
3509            printf("An error in find_WCET_input_from_prediction_model. A
solution was obtained with a binary variable that was not 0 and not 1.\n");
fflush(stdout); exit(-1);
3510          }
3511        }
3512      }
3513      *p_t = objval;
3514    } else {
```

```
3515      printf("An error in find_WCET_input_from_prediction_model. Did not
get an objective.\n"); fflush(stdout); exit(-1);
3516    }
3517  QUIT:
3518    if (error) { printf("ERROR: %s\n", GRBgeterrormsg(env)); exit(1); }
3519    GRBfreemodel(model);
3520    GRBfreeenv(env);
3521    freethememory_for_Gurobi_stuff();
3522  }
3523
3524  #define BATCH_SIZE 128
3525
3526  void fill_inputs_and_targets_with_synthetic_data(double* inputs,double*
targets,int n_examples) {
3527    int i; int j;
3528    for (int i = 0; i < BATCH_SIZE; i++) {
3529      for (int j = 0; j < inputsizeinnumberofbits;j++) {
3530        if ((random()%2)==1) {
3531          inputs[i*inputsizeinnumberofbits+j] = 1.0;
3532        } else {
3533          inputs[i*inputsizeinnumberofbits+j] = 0.0;
3534        }
3535      }
3536      targets[i] = 0.0;
3537      for (int j = 0; j < inputsizeinnumberofbits;j++) {
3538        if (inputs[i*inputsizeinnumberofbits+j]==1.0) {
3539          targets[i] = targets[i] + 0.0000001;
3540        } else {
3541          targets[i] = targets[i] - 0.0000001;
3542        }
3543      }
3544      targets[i] = targets[i] + 0.002;
3545    }
3546  }
3547
3548  void fill_inputs_and_targets_with_real_data(double* inputs,double*
targets,int n_examples) {
3549    int i; int j;
3550    int row_in_dataset;
3551    for (i=0;i<BATCH_SIZE;i++) {
3552      row_in_dataset = random()%n_examples;
3553      for (j=0;j<inputsizeinnumberofbits;j++) {
3554        if
(isbitonein_bitposition_in_integerarray(manyruns[row_in_dataset].inputtoprogr
am,j)) {
3555          inputs[i*inputsizeinnumberofbits+j] = 1.0;
3556        } else {
3557          inputs[i*inputsizeinnumberofbits+j] = 0.0;
3558        }
3559      }
3560      targets[i] = manyruns[row_in_dataset].t;
3561    }
3562  }
3563
3564
3565  int adding_this_row_would_cause_a_duplicate(int*
recording_of_id_of_examples_selected_in_batch,int i,int row_in_dataset) {
```

```
3566   int iterator;
3567   for (iterator=0;iterator<i;iterator++) {
3568     if
(recording_of_id_of_examples_selected_in_batch[iterator]==row_in_dataset) {
3569       return 1;
3570     }
3571   }
3572   return 0;
3573 }
3574
3575 void
fill_inputs_and_targets_with_real_data_using_uniform_sampling_over_execution_
time(double* inputs,double* targets,int n_examples) {
3576   int i; int j;
3577   int row_in_dataset;
3578   int* recording_of_id_of_examples_selected_in_batch;
3579   recording_of_id_of_examples_selected_in_batch =
malloc(BATCH_SIZE*sizeof(int)); if
(recording_of_id_of_examples_selected_in_batch==NULL) { printf("Memory
allocation failure in
fill_inputs_and_targets_with_real_data_using_uniform_sampling_over_execution_
time.\n"); exit(-1); }
3580   for (i=0;i<BATCH_SIZE;i++) {
3581     // In the code below, we check if a new row is already in the batch
and if so we reject it and get a new row instead
3582     // This avoids duplicate examples in a batch. I thought this
avoidance of duplicates would be beneficial for training
3583     // performance (more diversity leading to lower average training
error) but it turns out that the difference that it
3584     // makes is very small and not necessarily better.
3585     do {
3586       row_in_dataset =
obtain_row_using_uniform_sampling_over_execution_time(n_examples);
3587     } while
(adding_this_row_would_cause_a_duplicate(recording_of_id_of_examples_selected
_in_batch,i,row_in_dataset));
3588     recording_of_id_of_examples_selected_in_batch[i] = row_in_dataset;
3589     for (j=0;j<inputsizeinnumberofbits;j++) {
3590       if
(isbitonein_bitposition_in_integerarray(manyruns[row_in_dataset].inputtoprogr
am,j)) {
3591         inputs[i*inputsizeinnumberofbits+j] = 1.0;
3592       } else {
3593         inputs[i*inputsizeinnumberofbits+j] = 0.0;
3594       }
3595     }
3596     targets[i] = manyruns[row_in_dataset].t;
3597   }
3598   free(recording_of_id_of_examples_selected_in_batch);
3599 }
3600
3601 void NextBatchForTraining(TF_Tensor** inputs_tensor,
3602                           TF_Tensor** targets_tensor,
3603                           int n_examples) {
3604   double* inputs;
3605   double* targets;
```

```
3606   inputs  = malloc(BATCH_SIZE*inputsizeinnumberofbits*sizeof(double));
if (inputs==NULL) { printf("Memory allocation failure in
NextBatchForTraining. inputs.\n"); exit(-1); }
3607   targets = malloc(BATCH_SIZE*sizeof(double));
if (targets==NULL) { printf("Memory allocation failure in
NextBatchForTraining. targets.\n"); exit(-1); }
3608   //
fill_inputs_and_targets_with_synthetic_data(inputs,targets,n_examples);
3609   // fill_inputs_and_targets_with_real_data(inputs,targets,n_examples);
3610
fill_inputs_and_targets_with_real_data_using_uniform_sampling_over_execution_
time(inputs,targets,n_examples);
3611
3612   const int64_t dims_in[]  = {BATCH_SIZE, inputsizeinnumberofbits};
3613   const int64_t dims_out[] = {BATCH_SIZE, 1};
3614   size_t nbytes_in  = BATCH_SIZE *
sizeof(double)*inputsizeinnumberofbits;
3615   size_t nbytes_out = BATCH_SIZE * sizeof(double);
3616   *inputs_tensor = TF_AllocateTensor(TF_DOUBLE, dims_in, 2, nbytes_in);
3617   *targets_tensor = TF_AllocateTensor(TF_DOUBLE, dims_out, 2,
nbytes_out);
3618   memcpy(TF_TensorData(*inputs_tensor), inputs, nbytes_in);
3619   memcpy(TF_TensorData(*targets_tensor), targets, nbytes_out);
3620   free(inputs);
3621   free(targets);
3622 }
3623 #undef BATCH_SIZE
3624
3625 int ModelRunTrainStep(model_t* model,int n_examples) {
3626   TF_Tensor *x, *y;
3627   NextBatchForTraining(&x, &y, n_examples);
3628   TF_Output inputs[2] = {model->input, model->target};
3629   TF_Tensor* input_values[2] = {x, y};
3630   const TF_Operation* train_op[1] = {model->train_op};
3631   TF_SessionRun(model->session, NULL, inputs, input_values, 2,
3632                 /* No outputs */
3633                 NULL, NULL, 0, train_op, 1, NULL, model->status);
3634   TF_DeleteTensor(x);
3635   TF_DeleteTensor(y);
3636   return Okay(model->status);
3637 }
3638
3639 #define TRAINING_NUMBER_OF_BATCHES                          50000
3640 #define TRAINING_NUMBER_OF_BATCHES_INTERMEDIATE_REPORTING   12500
3641
3642 int find_WCET(int flag_read_executiontimemeasurements_from_files,int
flag_use_initialization_from_existing_files,char*
fn_with_executiontimemeasurements_inputs,char*
fn_with_executiontimemeasurements_times,char* fn_prefix,int programid,int
n_examples) {
3643   int i;
3644   char graph_def_filename[2000];
3645   char fn_to_use1[2000];
3646   char fn_to_use2[2000];
3647   char fn[2000];
3648   char fn2[2000];
3649   double t0; double t1; double t2;
```

```
3650    FILE* f;
3651    char cmd_to_run_python[2000];
3652    int samping_method_for_regression_with_affine_function;
3653
3654    sprintf(graph_def_filename,"%s_graph.pb",fn_prefix);
3655
3656    allocate_memory_for_regression_with_affine_function();
3657
3658
allocate_memory_for_predictedworstcaseinput_phase0_and_predictedworstcaseinpu
t_phase1();
3659
3660    allocate_memory_for_learnable_weights();
3661
3662    allocate_memory_for_manyruns_v2(         n_examples);
3663    index_of_examples = malloc(n_examples*sizeof(int)); if
(index_of_examples==NULL) { printf("Error in memory allocation for
index_of_examples.\n"); exit(-1); }
3664    if (flag_read_executiontimemeasurements_from_files) {
3665
read_captured_data_from_disk_two_explicit_fn(fn_with_executiontimemeasurement
s_inputs,fn_with_executiontimemeasurements_times,n_examples);
3666    } else {
3667      do_data_capture(programid,n_examples);
3668      printf("  Starting writing captured data to disk\n");
3669
write_captured_data_to_disk_two_explicit_fn(fn_with_executiontimemeasurements
_inputs,fn_with_executiontimemeasurements_times,n_examples);
3670    }
3671    calculate_min_max_average_executiontime_from_manyruns(n_examples);
3672
write_single_double_to_file(fn_prefix,"min_observed_time.txt",min_observed_ti
me);
3673
write_single_double_to_file(fn_prefix,"max_observed_time.txt",max_observed_ti
me);
3674
write_single_double_to_file(fn_prefix,"average_observed_time.txt",average_obs
erved_time);
3675    // print_min_max_average_executiontime_to_files(n_examples);
3676
3677    use_simulation_to_compute_weightconst_and_store_it(fn_prefix);
3678
3679    printf("n_examples = %d\n",n_examples);
3680
3681    fill_index_of_examples(n_examples);
3682
3683    samping_method_for_regression_with_affine_function =
SAMPLING_METHOD_FOR_REGRESSION_WITH_AFFINE_FUNCTION_UNIFORM_TIME;
3684    //
do_regression_with_affine_function(samping_method_for_regression_with_affine_
function,n_examples);
3685    //
write_weights_of_regression_with_affine_function_to_file(samping_method_for_r
egression_with_affine_function,fn_prefix);
```

```
3686
read_weights_of_regression_with_affine_function_from_file(samping_method_for_
regression_with_affine_function,fn_prefix);
3687
compute_prediction_and_error_for_all_examples_coefficients_from_learned_affin
e_function(n_examples);
3688
compute_statistics_based_on_regression_with_affine_function(n_examples);
3689
3690
use_simulation_to_compute_weightconst_to_be_used_for_w5_and_w6_and_store_it(f
n_prefix);
3691
3692    // compute_initialization_for_w5_and_w6_by_solving_MILP(fn_prefix);
3693
3694    // if (!flag_use_initialization_from_existing_files) {
3695      // do_regression_with_affine_function( n_examples); // this computes
a result that is used by write_initial_w0_to_file
3696      //
compute_statistics_based_on_regression_with_affine_function(n_examples);
3697      write_initial_w0_to_file( n_examples,fn_prefix);
3698      write_initial_w1_to_file( n_examples,fn_prefix);
3699      write_initial_w2_to_file( n_examples,fn_prefix);
3700      write_initial_w3_to_file( n_examples,fn_prefix);
3701      write_initial_w4_to_file( n_examples,fn_prefix);
3702      write_initial_b1_to_file( n_examples,fn_prefix);
3703      write_initial_b2_to_file( n_examples,fn_prefix);
3704      write_initial_b3_to_file( n_examples,fn_prefix);
3705      write_initial_b4_to_file( n_examples,fn_prefix);
3706
compute_initialization_for_w5_and_w6_by_solving_MILP(fn_prefix,n_examples);
3707      write_initial_w5_to_file( n_examples,fn_prefix); // this function
depends on a variable whose value is set when calling
compute_statistics_based_on_regression_with_affine_function
3708      write_initial_w6_to_file( n_examples,fn_prefix);
3709      write_initial_b56_to_file(n_examples,fn_prefix);
3710    // }
3711
3712    //
compute_and_print_statistics_on_prediction_and_error_of_training_using_affine
_model_based_on_w0_and_b56(fn_prefix,n_examples);
3713    //
print_statistics_on_prediction_and_error_of_training_using_affine_model_based
_on_w0_and_b56(fn_prefix,n_examples); // this assumes that
compute_statistics_based_on_regression_with_affine_function has been called
3714
print_statistics_on_prediction_and_error_of_training_using_affine_function(fn
_prefix,n_examples); // this assumes that
compute_statistics_based_on_regression_with_affine_function has been called
3715
3716    create_python_file(fn_prefix);
3717    sprintf(cmd_to_run_python,"python3 %s_model.py",fn_prefix);
3718    system(cmd_to_run_python);
3719
3720    model_t model;
3721    printf("Loading graph\n");
3722    if (!ModelCreate(&model, graph_def_filename)) return 1;
```

```
3723
3724    printf("Initializing model weights\n");
3725    if (!ModelInit(&model)) return 1;
3726
3727    i=0;
3728    fill_memory_with_learnable_weights(&model);
3729    print_all_weights_to_file(fn_prefix,i);
3730
3731    double* testdata;
3732    testdata = malloc(6*inputsizeinnumberofbits*sizeof(double)); if
(testdata==NULL) { printf("Memory allocation failure in NextBatchForTraining.
testdata.\n"); exit(-1); }
3733    fill_in_test_data(testdata,6,inputsizeinnumberofbits);
3734    printf("Initial predictions\n");
3735    if (!ModelPredict(&model, &testdata[0], 6)) return 1;
3736
3737    printf("Finding worst-case input. Before neural network training.\n");
3738    get_all_weights_from_file(fn_prefix,i);
3739
find_WCET_input_from_prediction_model(n_examples,fn_prefix,predictedworstcase
input,&predicted_time_predictedworstcaseinput);
3740
copy_inputtoprogram(predictedworstcaseinput_phase0,predictedworstcaseinput);
3741    predicted_time_predictedworstcaseinput_phase0 =
predicted_time_predictedworstcaseinput;
3742
3743    printf("Training for a few steps\n");
3744    i=0;
3745    while (i<TRAINING_NUMBER_OF_BATCHES) {
3746      if (i%TRAINING_NUMBER_OF_BATCHES_INTERMEDIATE_REPORTING==0) {
3747        printf("Obtain predictions and prediction errors %d\n",i);
fflush(stdout);
3748        if (!ModelPredictAllExamples(&model,(n_examples/10),fn_prefix,i))
return 1;
3749        if
(!ModelPredictAllExamples_specialized_for_uniform_time(&model,(n_examples/10)
,fn_prefix,i)) return 1;
3750        printf("Calculate statistics on predictions and prediction errors
%d\n",i); fflush(stdout);
3751
calculate_statistics_on_predictions_and_prediction_errors((n_examples/10),fn_
prefix,i);
3752        fill_memory_with_learnable_weights(&model);
3753        print_all_weights_to_file(fn_prefix,i);
3754      }
3755      if (!ModelRunTrainStep(&model,n_examples)) return 1;
3756      i++;
3757    }
3758    printf("Obtain predictions and prediction errors %d\n",i);
fflush(stdout);
3759    if (!ModelPredictAllExamples(&model,(n_examples/10),fn_prefix,i))
return 1;
3760    if
(!ModelPredictAllExamples_specialized_for_uniform_time(&model,(n_examples/10)
,fn_prefix,i)) return 1;
3761    printf("Calculate statistics on predictions and prediction errors
%d\n",i); fflush(stdout);
```

```
3762
calculate_statistics_on_predictions_and_prediction_errors((n_examples/10),fn_
prefix,i);
3763    fill_memory_with_learnable_weights(&model);
3764    print_all_weights_to_file(fn_prefix,i);
3765
3766    printf("Updated predictions\n");
3767    if (!ModelPredict(&model, &testdata[0], 6)) return 1;
3768
3769    printf("Finding worst-case input. After neural network training.\n");
3770    get_all_weights_from_file(fn_prefix,i);
3771
find_WCET_input_from_prediction_model(n_examples,fn_prefix,predictedworstcase
input,&predicted_time_predictedworstcaseinput);
3772    // printf(fn_to_use1,"%s_predictedworstcaseinput.dat",
fn_prefix);
3773    //
sprintf(fn_to_use2,"%s_predicted_time_predictedworstcaseinput.dat",fn_prefix)
;
3774    //
write_input_and_execution_time_to_files(fn_to_use1,fn_to_use2,predictedworstc
aseinput,predicted_time_predictedworstcaseinput);
3775
copy_inputtoprogram(predictedworstcaseinput_phase1,predictedworstcaseinput);
3776    predicted_time_predictedworstcaseinput_phase1 =
predicted_time_predictedworstcaseinput;
3777
3778    printf("Running with obtained worst-case input\n");
3779
3780    sleep(10);
3781    setprocessoraffinity_to_allow_just_a_single_processor_proc0();
3782    sleep(60);
3783
run_program_with_two_different_inputs_dont_use_filesystem_run_X_times(predict
edworstcaseinput_phase0,predictedworstcaseinput_phase1,programid,100,&t0,&t1)
;
3784
3785    sprintf(fn, "%s_0_predictedworstcaseinput.dat",        fn_prefix);
3786    sprintf(fn2,"%s_0_predictedworstcaseexecutiontime.dat",fn_prefix);
3787
write_input_and_execution_time_to_files(fn,fn2,predictedworstcaseinput_phase0
,t0);
3788
sprintf(fn,"%s_0_predictedworstcaseexecutiontime_notrun.dat",fn_prefix); f =
fopen(fn, "w+"); fprintf( f, "%18.12lf\n",
predicted_time_predictedworstcaseinput_phase0); fclose( f);
3789
3790    sprintf(fn, "%s_1_predictedworstcaseinput.dat",        fn_prefix);
3791    sprintf(fn2,"%s_1_predictedworstcaseexecutiontime.dat",fn_prefix);
3792
write_input_and_execution_time_to_files(fn,fn2,predictedworstcaseinput_phase1
,t1);
3793
sprintf(fn,"%s_1_predictedworstcaseexecutiontime_notrun.dat",fn_prefix); f =
fopen(fn, "w+"); fprintf( f, "%18.12lf\n",
predicted_time_predictedworstcaseinput_phase1); fclose( f);
3794
```

```
3795   setprocessoraffinity_to_allow_all_processors(); // we don't need to
restore this but it does not hurt
3796
3797   free(testdata);
3798
3799   free_memory_for_learnable_weights();
3800   free_memory_for_manyruns_v2(n_examples);
3801   free(index_of_examples);
3802
3803
free_memory_for_predictedworstcaseinput_phase0_and_predictedworstcaseinput_ph
ase1();
3804
3805   free_memory_for_regression_with_affine_function();
3806 }
3807
3808 void copy_example(int dest_index,int source_index) {
3809   int k;
3810
copy_inputtoprogram(manyruns[dest_index].inputtoprogram,manyruns[source_index
].inputtoprogram);
3811   manyruns[dest_index].t = manyruns[source_index].t;
3812   manyruns[dest_index].error = manyruns[source_index].error;
3813   for
(k=0;k<REPLICATION_TO_GET_MEDIAN_OF_EXECUTIONTIME_MEASUREMENTS;k++) {
3814     manyruns[dest_index].recordedexecutiontimes[k] =
manyruns[source_index].recordedexecutiontimes[k];
3815   }
3816   manyruns[dest_index].prediction = manyruns[source_index].prediction;
3817 }
3818
3819 int cmpfunc_sort_examples(const void * a, const void * b) {
3820   struct onerun* p_a;
3821   struct onerun* p_b;
3822   p_a = (struct onerun*) a;
3823   p_b = (struct onerun*) b;
3824   if ((p_a->t) < (p_b->t)) {
3825     return -1;
3826   } else {
3827     if ((p_a->t) == (p_b->t)) {
3828       return 0;
3829     } else {
3830       return 1;
3831     }
3832   }
3833 }
3834
3835 void sort_all_examples(int n_examples_in_file) {
3836   qsort(manyruns, n_examples_in_file, sizeof(struct onerun),
cmpfunc_sort_examples);
3837 }
3838
3839 void trim_examples(int n_examples_in_file,int n_examples,int
n_examples_part1,int n_examples_part2,int n_examples_part3) {
3840   int source_index; int dest_index; double source_index_double; double
step;
```

```
3841   step = (n_examples_in_file-n_examples_part1-
n_examples_part3)/((double) n_examples_part2);
3842   dest_index = n_examples_part1;
3843   source_index_double = n_examples_part1;
3844   source_index = source_index_double;
3845   while (dest_index<n_examples_part1+n_examples_part2) {
3846     if (dest_index!=source_index) {
3847       copy_example(dest_index,source_index);
3848     }
3849     dest_index = dest_index + 1;
3850     source_index_double = source_index_double + step;
3851     source_index = source_index_double;
3852   }
3853   dest_index = n_examples_part1+n_examples_part2;
3854   source_index = n_examples_in_file-n_examples_part3;
3855   while (dest_index<n_examples_part1+n_examples_part2+n_examples_part3)
{
3856     if (dest_index!=source_index) {
3857       copy_example(dest_index,source_index);
3858     }
3859     dest_index = dest_index + 1;
3860     source_index = source_index + 1;
3861   }
3862 }
3863
3864 void fillobj_for_objective_function_learn_w0_b56_through_MILP() {
3865   int j;
3866   j = 0;
3867   obj[j]   = 1.0;
3868   lb[j]    = 0.0;
3869   ub[j]    = GRB_INFINITY;
3870   vtype[j] = GRB_CONTINUOUS;
3871   for (j=1;j<nelements_of_obj;j++) {
3872     obj[j]   = 0.0;
3873     lb[j]    = -GRB_INFINITY;
3874     ub[j]    = GRB_INFINITY;
3875     vtype[j] = GRB_CONTINUOUS;
3876   }
3877 }
3878
3879 // double obtain_normalized_execution_time_from_execution_time(double t)
{
3880 //   double mint; double maxt; double averaget;
3881 //   get_min_max_average_execution_time(&mint,&maxt,&averaget);
3882 //   return (t-mint)/(maxt-mint);
3883 // }
3884
3885 void
setup_datastructures_for_specific_learn_w0_b56_through_MILP_leq_constraint(in
t i) {
3886   int j;
3887   double mint; double maxt; double averaget;
3888   nelements_of_constr = 0;
3889   ind[nelements_of_constr] = 0;
3890   val[nelements_of_constr] = -1.0;
3891   nelements_of_constr = nelements_of_constr + 1;
3892   for (j=0;j<inputsizeinnumberofbits;j++) {
```

```
3893      if
(isbitonein_bitposition_in_integerarray(manyruns[i].inputtoprogram,j)) {
3894        ind[nelements_of_constr] = 1+j;
3895        val[nelements_of_constr] = 1.0;
3896        nelements_of_constr = nelements_of_constr + 1;
3897      }
3898    }
3899    ind[nelements_of_constr] = inputsizeinnumberofbits+1;
3900    val[nelements_of_constr] = 1.0;
3901    nelements_of_constr = nelements_of_constr + 1;
3902    // rhs = obtain_normalized_execution_time_from_execution_time(
manyruns[i].t);
3903    // rhs = obtain_normalized_execution_time_from_execution_time(
manyruns[i].t) * 100.0;
3904    get_min_max_average_execution_time(&mint,&maxt,&averaget);
3905    rhs = (manyruns[i].t-mint)/(maxt-mint) * 100.0;
3906  }
3907
3908  void
setup_datastructures_for_specific_learn_w0_b56_through_MILP_flip_from_leq_to_
geq() {
3909    val[0] = val[0] * (-1.0);
3910  }
3911
3912  void write_sol_from_Gurobi_to_textfile_init_z(char* fn_prefix) {
3913    char fn[200]; FILE* f;
3914    double mint; double maxt; double averaget;
3915    get_min_max_average_execution_time(&mint,&maxt,&averaget);
3916    sprintf(fn,"%s_init_z.txt",fn_prefix);
3917    f = fopen(fn, "w+" ); if (f==NULL) { printf("Error in
write_sol_from_Gurobi_to_textfile_init_z\n"); fflush(stdout); exit(-1); }
3918    // fprintf(f,"%30.25lf\n", sol[0]);
3919    fprintf(f,"%30.25lf\n", (sol[0]/100.0)*(maxt-mint));
3920    fclose(f);
3921  }
3922  void write_sol_from_Gurobi_to_textfile_init_w0(char* fn_prefix) {
3923    char fn[200]; FILE* f; int j;
3924    double mint; double maxt; double averaget;
3925    get_min_max_average_execution_time(&mint,&maxt,&averaget);
3926    sprintf(fn,"%s_init_w0.txt",fn_prefix);
3927    f = fopen(fn, "w+" ); if (f==NULL) { printf("Error in
write_sol_from_Gurobi_to_textfile_init_w0\n"); fflush(stdout); exit(-1); }
3928    for (j=0;j<inputsizeinnumberofbits;j++) {
3929      // fprintf(f,"%30.25lf\n", sol[j+1]);
3930      fprintf(f,"%30.25lf\n", (sol[j+1]/100.0)*(maxt-mint));
3931    }
3932    fclose(f);
3933  }
3934  void write_sol_from_Gurobi_to_textfile_init_b56(char* fn_prefix) {
3935    char fn[200]; FILE* f;
3936    double mint; double maxt; double averaget;
3937    get_min_max_average_execution_time(&mint,&maxt,&averaget);
3938    sprintf(fn,"%s_init_b56.txt",fn_prefix);
3939    f = fopen(fn, "w+" ); if (f==NULL) { printf("Error in
write_sol_from_Gurobi_to_textfile_init_b56\n"); fflush(stdout); exit(-1); }
3940    // fprintf(f,"%30.25lf\n", sol[inputsizeinnumberofbits+1]);
```

```
3941   fprintf(f,"%30.25lf\n",
mint+((sol[inputsizeinnumberofbits+1])/100.0)*(maxt-mint));
3942   fclose(f);
3943 }
3944 void write_sol_from_Gurobi_to_textfiles_z_w0_b56(char* fn_prefix) {
3945   write_sol_from_Gurobi_to_textfile_init_z(fn_prefix);
3946   write_sol_from_Gurobi_to_textfile_init_w0(fn_prefix);
3947   write_sol_from_Gurobi_to_textfile_init_b56(fn_prefix);
3948 }
3949
3950 void read_w0_from_init_textfile(char* fn_prefix) {
3951   char fn[200]; FILE* f; int j;
3952   sprintf(fn,"%s_init_w0.txt",fn_prefix);
3953   f = fopen(fn, "r" ); if (f==NULL) { printf("Error in
read_w0_from_init_textfile\n"); fflush(stdout); exit(-1); }
3954   for (j=0;j<inputsizeinnumberofbits;j++) {
3955     fscanf(f,"%lf", &(p_w0_value[j]));
3956   }
3957   fclose(f);
3958 }
3959 void read_b56_from_init_textfile(char* fn_prefix) {
3960   char fn[200]; FILE* f;
3961   sprintf(fn,"%s_init_b56.txt",fn_prefix);
3962   f = fopen(fn, "r" ); if (f==NULL) { printf("Error in
read_b56_from_init_textfile\n"); fflush(stdout); exit(-1); }
3963   fscanf(f,"%lf", p_b56_value);
3964   fclose(f);
3965 }
3966 void read_w0_b56_from_init_textfile(char* fn_prefix) {
3967   read_w0_from_init_textfile(fn_prefix);
3968   read_b56_from_init_textfile(fn_prefix);
3969 }
3970
3971 // indices in Gurobi for each variable
3972 //   z   is index 0
3973 //   w_0 is index 1
3974 //   w_1 is index 2
3975 //   ...
3976 //   w_{N-1} is index N
3977 //   w_{N}   is index b56
3978
3979 // should we introduce variables for pred^i ???
3980
3981
3982 // Use OptimalityTol
3983 //   the default is 10^{-6}
3984 //   we should change it to:
3985 //      (maxt-mint)*10^{-4}
3986 //   for example:
3987 //     for bubblesort, we will get OptimalityTol=(2.7ms-1.99ms)*10^{-4}
= 0.8  * 10^{-7}=0.080 us=0.000000080
3988 //     for heapsort,   we will get OptimalityTol=(1.04ms-0.88ms)*10^{-4}
= 0.16 * 10^{-7}=0.016 us=0.000000016
3989 //   Maybe there is no advantage in setting OptimalityTol to these
values. The reason is that it is only advantagesous if we can use larger
number
```

```
3990 //   of training examples and then OptimalityTol allows that. However,
if we have larger number of training examples, then we run out of memory
anyway.
3991
3992 // this assumes that examples are sorted in ascending order
3993 double get_value_of_optimality_tol(int n_examples) {
3994   double computed_value_of_optimality_tol;
3995   // printf("manyruns[n_examples-1].t = %30.25lf\n",
manyruns[n_examples-1].t);
3996   // printf("manyruns[0].t            = %30.25lf\n", manyruns[0].t
);
3997   computed_value_of_optimality_tol = (manyruns[n_examples-1].t -
manyruns[0].t)*0.0001;
3998   // printf("computed_value_of_optimality_tol = %30.25lf\n",
computed_value_of_optimality_tol);
3999   return computed_value_of_optimality_tol;
4000 }
4001
4002 void learn_w0_b56_through_MILP(int n_examples,char* fn_prefix) {
4003   int i;
4004   char gurobi_learn_w0_b56_through_MILP_log_file[200];
4005   char gurobi_learn_w0_b56_through_MILP_lp_file[200];
4006   char gurobi_learn_w0_b56_through_MILP_sol_file[200];
4007
sprintf(gurobi_learn_w0_b56_through_MILP_log_file,"%s_learn_w0_b56_through_MI
LP.log",fn_prefix);
4008   sprintf(gurobi_learn_w0_b56_through_MILP_lp_file,
"%s_learn_w0_b56_through_MILP.lp", fn_prefix);
4009
sprintf(gurobi_learn_w0_b56_through_MILP_sol_file,"%s_learn_w0_b56_through_MI
LP.sol",fn_prefix);
4010   nelements_of_obj = 1+inputsizeinnumberofbits+1;
4011   dothememoryallocation_for_Gurobi_stuff();
4012   error = GRBloadenv(&env, gurobi_learn_w0_b56_through_MILP_log_file);
4013   if (error) goto QUIT;
4014   error = GRBsetintparam(env, GRB_INT_PAR_METHOD, GRB_METHOD_BARRIER);
4015   if (error) goto QUIT;
4016   // error = GRBsetdblparam(env, GRB_DBL_PAR_OPTIMALITYTOL,
get_value_of_optimality_tol(n_examples));
4017   // if (error) goto QUIT;
4018   fillobj_for_objective_function_learn_w0_b56_through_MILP();
4019   error = GRBnewmodel(env, &model, "learn_w0_b56_through_MILP",
nelements_of_obj, obj, lb, ub, vtype, NULL);
4020   if (error) goto QUIT;
4021   error = GRBsetintattr( model, GRB_INT_ATTR_MODELSENSE, GRB_MINIMIZE);
4022   if (error) goto QUIT;
4023   error = GRBupdatemodel( model);
4024   if (error) goto QUIT;
4025   printf("Start add constraints\n"); fflush(stdout);
4026   for (i=0;i<n_examples;i++) {
4027
setup_datastructures_for_specific_learn_w0_b56_through_MILP_leq_constraint(i)
;
4028     error = GRBaddconstr( model, nelements_of_constr, ind, val,
GRB_LESS_EQUAL, rhs, NULL );
4029     if (error) goto QUIT;
```

```
4030
setup_datastructures_for_specific_learn_w0_b56_through_MILP_flip_from_leq_to_
geq();
4031      error = GRBaddconstr( model, nelements_of_constr, ind, val,
GRB_GREATER_EQUAL, rhs, NULL );
4032      if (error) goto QUIT;
4033    }
4034    printf("Finish add constraints\n"); fflush(stdout);
4035    error = GRBupdatemodel( model);
4036    if (error) goto QUIT;
4037    // error = GRBwrite( model,gurobi_learn_w0_b56_through_MILP_lp_file);
4038    // if (error) goto QUIT;
4039    error = GRBoptimize(model);
4040    if (error) goto QUIT;
4041    error = GRBupdatemodel( model);
4042    if (error) goto QUIT;
4043    error = GRBgetintattr( model, GRB_INT_ATTR_STATUS, &optimstatus);
4044    if (error) goto QUIT;
4045    if (optimstatus == GRB_OPTIMAL) {
4046      error = GRBwrite( model,gurobi_learn_w0_b56_through_MILP_sol_file);
4047      if (error) goto QUIT;
4048      error = GRBgetdblattr( model, GRB_DBL_ATTR_OBJVAL, &objval);
4049      if (error) goto QUIT;
4050      error = GRBgetdblattrarray( model, GRB_DBL_ATTR_X, 0,
nelements_of_obj, sol);
4051      if (error) goto QUIT;
4052      write_sol_from_Gurobi_to_textfiles_z_w0_b56(fn_prefix);
4053    } else {
4054      printf("An error in learn_w0_b56_through_MILP. Did not get an
objective.\n"); fflush(stdout); exit(-1);
4055    }
4056  QUIT:
4057    if (error) { printf("ERROR: %s\n", GRBgeterrormsg(env)); exit(1); }
4058    GRBfreemodel(model);
4059    GRBfreeenv(env);
4060    freethememory_for_Gurobi_stuff();
4061    printf("Finish learn_w0_b56_through_MILP\n"); fflush(stdout);
4062  }
4063
4064
4065  void compute_error_and_prediction_based_on_w0_b56_for_one_example(int i)
{
4066    int j;
4067    manyruns[i].prediction = p_b56_value[0];
4068    for (j=0;j<inputsizeinnumberofbits;j++) {
4069      if
(isbitonein_bitposition_in_integerarray(manyruns[i].inputtoprogram,j)) {
4070        manyruns[i].prediction = manyruns[i].prediction + p_w0_value[j];
4071      }
4072    }
4073    manyruns[i].error = manyruns[i].prediction - manyruns[i].t;
4074  }
4075
4076  struct myargstruct3 {
4077    int loindex;
4078    int hiindex;
4079  };
```

```
4080 pthread_t threads3[NUM_THREADS_USED_FOR_WORK];
4081 pthread_attr_t attrs3[NUM_THREADS_USED_FOR_WORK];
4082 struct myargstruct3 myargstruct_array3[NUM_THREADS_USED_FOR_WORK];
4083
4084 void set_indices_in_myargstruct_array3(int th,int n_examples) {
4085   int objects_per_worker;
4086   if (n_examples%NUM_THREADS_USED_FOR_WORK==0) {
4087     objects_per_worker = n_examples/NUM_THREADS_USED_FOR_WORK;
4088   } else {
4089     objects_per_worker = n_examples/NUM_THREADS_USED_FOR_WORK + 1;
4090   }
4091   myargstruct_array3[th].loindex = th    * objects_per_worker;
4092   myargstruct_array3[th].hiindex = (th+1)* objects_per_worker - 1;
4093   if (myargstruct_array3[th].hiindex>n_examples-1) {
4094     myargstruct_array3[th].hiindex = n_examples-1;
4095   }
4096 }
4097
4098 __attribute__((optimize("-O3"))) void* worker3(void *p) {
4099   int i;
4100   for (i=((struct myargstruct3*) p)->loindex;i<=((struct myargstruct3*)
p)->hiindex;i++) {
4101     compute_error_and_prediction_based_on_w0_b56_for_one_example(i);
4102   }
4103   pthread_exit(NULL);
4104 }
4105
4106 void compute_error_and_prediction_based_on_w0_b56(int n_examples) {
4107   int rc; int th; void* status;
4108   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
4109      pthread_attr_init(&(attrs3[th]));
4110      pthread_attr_setdetachstate(&(attrs3[th]),PTHREAD_CREATE_JOINABLE);
4111      set_indices_in_myargstruct_array3(th,n_examples);
4112      rc = pthread_create(&(threads3[th]),&(attrs3[th]),worker3,(void*)
(&(myargstruct_array3[th])));
4113      if (rc){
4114         printf("ERROR; return code from pthread_create() is %d\n", rc);
4115         exit(-1);
4116      }
4117   }
4118   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
4119      rc = pthread_join(threads3[th], &status);
4120      if (rc) {
4121         printf("ERROR; return code from pthread_join() is %d\n", rc);
4122         exit(-1);
4123      }
4124   }
4125   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
4126     pthread_attr_destroy( &(attrs3[th]) );
4127   }
4128 }
4129
4130 void write_all_predictions_to_file(char* fn_prefix, int n_examples) {
4131   char fn[200]; FILE* f; int i;
4132   sprintf(fn,"%s_all_predictions",fn_prefix);
4133   f = fopen(fn, "w+");
```

```
4134   if (f==NULL) { printf("In write_all_predictions_to_file. Opening file.
failed\n"); exit(-1); }
4135   for (i=0;i<n_examples;i++) {
4136     fwrite(&(manyruns[i].prediction),sizeof(double),1,f);
4137   }
4138   fclose( f);
4139 }
4140 void write_all_errors_to_file(char* fn_prefix, int n_examples) {
4141   char fn[200]; FILE* f; int i;
4142   sprintf(fn,"%s_all_errors",fn_prefix);
4143   f = fopen(fn, "w+");
4144   if (f==NULL) { printf("In write_all_errors_to_fileOpening file.
failed\n"); exit(-1); }
4145   for (i=0;i<n_examples;i++) {
4146     fwrite(&(manyruns[i].error),sizeof(double),1,f);
4147   }
4148   fclose( f);
4149 }
4150
4151 void read_all_predictions_from_file(char* fn_prefix, int n_examples) {
4152   char fn[200]; FILE* f; int i;
4153   sprintf(fn,"%s_all_predictions",fn_prefix);
4154   f = fopen(fn, "r");
4155   if (f==NULL) { printf("In read_all_predictions_to_file. Opening file.
failed\n"); exit(-1); }
4156   for (i=0;i<n_examples;i++) {
4157     fread(&(manyruns[i].prediction),sizeof(double),1,f);
4158   }
4159   fclose( f);
4160 }
4161 void read_all_errors_from_file(char* fn_prefix, int n_examples) {
4162   char fn[200]; FILE* f; int i;
4163   sprintf(fn,"%s_all_errors",fn_prefix);
4164   f = fopen(fn, "r");
4165   if (f==NULL) { printf("In read_all_errors_to_file. Opening file.
failed\n"); exit(-1); }
4166   for (i=0;i<n_examples;i++) {
4167     fread(&(manyruns[i].error),sizeof(double),1,f);
4168   }
4169   fclose( f);
4170 }
4171
4172 int get_number_of_examples_with_abs_error_exceeding(int
n_examples,double threshold) {
4173   int count; int i;
4174   count = 0;
4175   for (i=0;i<n_examples;i++) {
4176     if (fabs(manyruns[i].error)>threshold) {
4177       count = count + 1;
4178     }
4179   }
4180   return count;
4181 }
4182
4183 void write_histogram_of_abs_error(char* fn_prefix,char*
fn_regression_with_affine_function_histogram_abs_error,int n_examples,double
regression_with_affine_function_max_abs_error) {
```

```
4184   char fn[200]; FILE* f; int i; double threshold; int count;
4185
sprintf(fn,"%s_%s",fn_prefix,fn_regression_with_affine_function_histogram_abs
_error);
4186   f = fopen(fn, "w+");
4187   // if (f!=NULL) { printf("Opening file. success\n"); } else {
printf("Opening file. failed\n"); }
4188   if (f==NULL) { printf("In write_histogram_of_abs_error. Opening file.
failed\n"); exit(-1); }
4189   for (i=1;i<=100;i++) {
4190     threshold =
regression_with_affine_function_max_abs_error*((double)i)/100.0;
4191     count =
get_number_of_examples_with_abs_error_exceeding(n_examples,threshold);
4192     fprintf(f,"%lf %d\n",threshold,count);
4193   }
4194   fclose( f);
4195 }
4196
4197 void write_summary_of_predictions_and_errors_to_files(char*
fn_prefix,int n_examples) {
4198
compute_statistics_based_on_regression_with_affine_function_based_on_all_exam
ples(n_examples);
4199
write_single_double_to_file(fn_prefix,"regression_with_affine_function_min_pr
ediction.txt",regression_with_affine_function_min_prediction);
4200
write_single_double_to_file(fn_prefix,"regression_with_affine_function_max_pr
ediction.txt",regression_with_affine_function_max_prediction);
4201
write_single_double_to_file(fn_prefix,"regression_with_affine_function_max_ab
s_error.txt",regression_with_affine_function_max_abs_error);
4202
write_single_int_to_file(fn_prefix,"regression_with_affine_function_index_wit
h_max_abs_error.txt",regression_with_affine_function_index_with_max_abs_error
);
4203
write_single_double_to_file(fn_prefix,"regression_with_affine_function_predic
tion_of_example_with_max_abs_error.txt",
4204
regression_with_affine_function_prediction_of_example_with_max_abs_error);
4205
write_single_double_to_file(fn_prefix,"regression_with_affine_function_t_of_e
xample_with_max_abs_error.txt",
4206     regression_with_affine_function_t_of_example_with_max_abs_error);
4207
write_single_double_to_file(fn_prefix,"regression_with_affine_function_averag
e_abs_error.txt",regression_with_affine_function_average_abs_error);
4208
write_histogram_of_abs_error(fn_prefix,"regression_with_affine_function_histo
gram_abs_error.txt",n_examples,regression_with_affine_function_max_abs_error)
;
4209 }
4210
4211 void obtain_trimmed_data_fn(char* fn_trimmed_data,char* fn) {
4212   sprintf(fn_trimmed_data,"%s_trimmed",fn);
```

```
4213 }
4214
4215 int cmpfunc_sort_abs_error_index(const void * a, const void * b) {
4216   int index_a; int index_b;
4217   double abs_error_a; double abs_error_b;
4218   index_a = *((int*)a);
4219   index_b = *((int*)b);
4220   abs_error_a = fabs(manyruns[index_a].error);
4221   abs_error_b = fabs(manyruns[index_b].error);
4222   if (abs_error_a < abs_error_b) {
4223     return -1;
4224   } else {
4225     if (abs_error_a == abs_error_b) {
4226       return 0;
4227     } else {
4228       return 1;
4229     }
4230   }
4231 }
4232
4233 void update_abs_error_index(int n_examples_in_file) {
4234   int i;
4235   for (i=0;i<n_examples_in_file;i++) {
4236     abs_error_index_of_examples[i] = i;
4237   }
4238   qsort(abs_error_index_of_examples, n_examples_in_file, sizeof(int),
cmpfunc_sort_abs_error_index);
4239 }
4240
4241 int is_this_example_selected(int i) {
4242   int iterator;
4243   iterator = 0;
4244   while (iterator<n_selected_index_of_examples) {
4245     if (selected_index_of_examples[iterator]==i) {
4246       return 1;
4247     }
4248     iterator++;
4249   }
4250   return 0;
4251 }
4252
4253 void print_selected_index_of_examples(char* fn_prefix,int phase) {
4254   char fn_to_use[2000]; FILE* f; int iterator;
4255   sprintf(fn_to_use,"%s_selected_index_of_examples_%d",fn_prefix,phase);
4256   f = fopen(fn_to_use, "w" ); if (f==NULL) { printf("Error
print_selected_index_of_examples %s\n",fn_to_use); exit(-1); }
4257   for (iterator=0;iterator<n_selected_index_of_examples;iterator++) {
4258     fprintf(f,"%d %lf\n",selected_index_of_examples[iterator],
manyruns[selected_index_of_examples[iterator]].t );
4259   }
4260   fclose( f);
4261 }
4262
4263 void learn_w0_b56_through_MILP_many_phases(int n_examples,int
n_examples_in_file,char* fn_prefix) {
4264   int i; int phase; int n_lo_examples; int n_hi_examples; int
target_n_selected_index_of_examples; int index_to_large_abs_error;
```

```
4265    char gurobi_learn_w0_b56_through_MILP_log_file[200];
4266    char gurobi_learn_w0_b56_through_MILP_lp_file[200];
4267    char gurobi_learn_w0_b56_through_MILP_sol_file[200];
4268    char gurobi_learn_w0_b56_through_MILP_sol_file_phase_specific[200];
4269
sprintf(gurobi_learn_w0_b56_through_MILP_log_file,"%s_learn_w0_b56_through_MI
LP_many_phases.log",fn_prefix);
4270    sprintf(gurobi_learn_w0_b56_through_MILP_lp_file,
"%s_learn_w0_b56_through_MILP_many_phases.lp", fn_prefix);
4271
sprintf(gurobi_learn_w0_b56_through_MILP_sol_file,"%s_learn_w0_b56_through_MI
LP_many_phases.sol",fn_prefix);
4272    nelements_of_obj = 1+inputsizeinnumberofbits+1;
4273    dothememoryallocation_for_Gurobi_stuff();
4274    error = GRBloadenv(&env, gurobi_learn_w0_b56_through_MILP_log_file);
4275    if (error) goto QUIT;
4276    error = GRBsetintparam(env, GRB_INT_PAR_METHOD, GRB_METHOD_BARRIER);
// we set it to barrier because otherwise, it will try all three methods and
this consumes too much memory
4277    if (error) goto QUIT;
4278    fillobj_for_objective_function_learn_w0_b56_through_MILP();
4279    error = GRBnewmodel(env, &model,
"learn_w0_b56_through_MILP_many_phases", nelements_of_obj, obj, lb, ub,
vtype, NULL);
4280    if (error) goto QUIT;
4281    error = GRBsetintattr( model, GRB_INT_ATTR_MODELSENSE, GRB_MINIMIZE);
4282    if (error) goto QUIT;
4283    error = GRBupdatemodel( model);
4284    if (error) goto QUIT;
4285    phase = 0;
4286    n_selected_index_of_examples = 0;
4287    n_lo_examples = (1+inputsizeinnumberofbits)/2;
4288    n_hi_examples  = (1+inputsizeinnumberofbits) - n_lo_examples;
4289    while (phase<N_PHASES_ITERATIONS_AFFINE_MODEL) {
4290      target_n_selected_index_of_examples = (1+inputsizeinnumberofbits) +
4291        (phase+1)*((double) (n_examples-
(1+inputsizeinnumberofbits)))/N_PHASES_ITERATIONS_AFFINE_MODEL;
4292      if (target_n_selected_index_of_examples>n_examples) {
target_n_selected_index_of_examples = n_examples; }
4293      if (phase>0) { index_to_large_abs_error = n_examples_in_file-1; }
4294      while
(n_selected_index_of_examples<target_n_selected_index_of_examples) {
4295        if (n_selected_index_of_examples<n_lo_examples) {
4296          i = index_of_examples[n_selected_index_of_examples];
4297        } else {
4298          if (n_selected_index_of_examples<n_lo_examples+n_hi_examples) {
4299            i = index_of_examples[n_examples_in_file-1-
(n_selected_index_of_examples-n_lo_examples)];
4300          } else {
4301            if (phase==0) {
4302              i =
obtain_row_using_uniform_sampling_given_lb_index_and_ub_index_of_sorted_order
(n_lo_examples,n_examples_in_file-n_hi_examples-1);
4303              while (is_this_example_selected(i)) {
4304                i =
obtain_row_using_uniform_sampling_given_lb_index_and_ub_index_of_sorted_order
(n_lo_examples,n_examples_in_file-n_hi_examples-1);
```

```
4305              }
4306            } else {
4307              i = abs_error_index_of_examples[index_to_large_abs_error];
4308              while (is_this_example_selected(i)) {
4309                index_to_large_abs_error = index_to_large_abs_error - 1;
4310                if (index_to_large_abs_error<0) { printf("An error has
occurred in learn_w0_b56_through_MILP_many_phases\n"); exit(-1); }
4311                i = abs_error_index_of_examples[index_to_large_abs_error];
4312              }
4313              index_to_large_abs_error = index_to_large_abs_error - 1;
4314            }
4315          }
4316        }
4317
setup_datastructures_for_specific_learn_w0_b56_through_MILP_leq_constraint(i)
;
4318        error = GRBaddconstr( model, nelements_of_constr, ind, val,
GRB_LESS_EQUAL, rhs, NULL );
4319        if (error) goto QUIT;
4320
setup_datastructures_for_specific_learn_w0_b56_through_MILP_flip_from_leq_to_
geq();
4321        error = GRBaddconstr( model, nelements_of_constr, ind, val,
GRB_GREATER_EQUAL, rhs, NULL );
4322        if (error) goto QUIT;
4323        selected_index_of_examples[n_selected_index_of_examples] = i;
4324        n_selected_index_of_examples = n_selected_index_of_examples + 1;
4325      }
4326      print_selected_index_of_examples(fn_prefix,phase);
4327      error = GRBupdatemodel( model);
4328      if (error) goto QUIT;
4329      // error = GRBwrite(
model,gurobi_learn_w0_b56_through_MILP_lp_file);
4330      // if (error) goto QUIT;
4331      error = GRBoptimize(model);
4332      if (error) goto QUIT;
4333      error = GRBupdatemodel( model);
4334      if (error) goto QUIT;
4335      error = GRBgetintattr( model, GRB_INT_ATTR_STATUS, &optimstatus);
4336      if (error) goto QUIT;
4337      if (optimstatus == GRB_OPTIMAL) {
4338
sprintf(gurobi_learn_w0_b56_through_MILP_sol_file_phase_specific,"%s_learn_w0
_b56_through_MILP_many_phases_phase_specific_%d.sol",fn_prefix,phase);
4339        error = GRBwrite(
model,gurobi_learn_w0_b56_through_MILP_sol_file);
4340        error = GRBwrite(
model,gurobi_learn_w0_b56_through_MILP_sol_file_phase_specific);
4341        if (error) goto QUIT;
4342        error = GRBgetdblattr( model, GRB_DBL_ATTR_OBJVAL, &objval);
4343        if (error) goto QUIT;
4344        error = GRBgetdblattrarray( model, GRB_DBL_ATTR_X, 0,
nelements_of_obj, sol);
4345        if (error) goto QUIT;
4346        write_sol_from_Gurobi_to_textfiles_z_w0_b56(fn_prefix); // later
on, we may want to change this code to reflect that we are normalizing
4347        read_w0_b56_from_init_textfile(fn_prefix);
```

```
4348        compute_error_and_prediction_based_on_w0_b56(n_examples_in_file);
4349        update_abs_error_index(n_examples_in_file);
4350      } else {
4351        printf("An error in learn_w0_b56_through_MILP_many_phases. Did not
get an objective.\n"); fflush(stdout); exit(-1);
4352      }
4353      phase = phase + 1;
4354    }
4355  QUIT:
4356    if (error) { printf("ERROR: %s\n", GRBgeterrormsg(env)); exit(1); }
4357    GRBfreemodel(model);
4358    GRBfreeenv(env);
4359    freethememory_for_Gurobi_stuff();
4360    printf("Finish learn_w0_b56_through_MILP_many_phases\n");
fflush(stdout);
4361  }
4362
4363  int sorted_order_of_variables_in_affine_model[inputsizeinnumberofbits];
4364
4365  int cmpfunc_sort_order_of_variables_in_affine_model(const void * a,
const void * b) {
4366    int* p_a;
4367    int* p_b;
4368    double abs_coefficient_a;
4369    double abs_coefficient_b;
4370    p_a = (int*)a;
4371    p_b = (int*)b;
4372    abs_coefficient_a = fabs(p_w0_value[*p_a]);
4373    abs_coefficient_b = fabs(p_w0_value[*p_b]);
4374    if (abs_coefficient_a < abs_coefficient_b) {
4375      return -1;
4376    } else {
4377      if (abs_coefficient_a == abs_coefficient_b) {
4378        return 0;
4379      } else {
4380        return 1;
4381      }
4382    }
4383  }
4384
4385  void fill_in_sorted_order_of_variables_in_affine_model() {
4386    int iterator;
4387    for (iterator=0;iterator<inputsizeinnumberofbits;iterator++) {
4388      sorted_order_of_variables_in_affine_model[iterator] = iterator;
4389    }
4390    qsort(sorted_order_of_variables_in_affine_model,
inputsizeinnumberofbits, sizeof(int),
cmpfunc_sort_order_of_variables_in_affine_model);
4391  }
4392
4393  void
fillobj_for_objective_function_improve_learning_affine_prediction_model() {
4394    int j;
4395    j = 0;
4396    obj[j]    = 1.0;
4397    lb[j]     = 0.0;
4398    ub[j]     = GRB_INFINITY;
```

```
4399   vtype[j] = GRB_CONTINUOUS;
4400   for (j=1;j<nelements_of_obj;j++) {
4401     obj[j]   = 0.0;
4402     lb[j]    = -GRB_INFINITY;
4403     ub[j]    = GRB_INFINITY;
4404     vtype[j] = GRB_CONTINUOUS;
4405   }
4406 }
4407
4408 // #define N_LARGE_ITERATIONS 40
4409 // #define N_ELEMENTS_IN_J_ARRAY 25
4410 // #define N_LARGE_ITERATIONS 50
4411 // #define N_ELEMENTS_IN_J_ARRAY 20 // during sifting, we end up with 1%
CPU utilization; that is bad.
4412 #define N_LARGE_ITERATIONS 100
4413 #define N_ELEMENTS_IN_J_ARRAY 10
4414
4415 // If N_ELEMENTS_IN_J_ARRAY is higher, then we end up with swapping on
my computer (256GB of DRAM).
4416 // With N_ELEMENTS_IN_J_ARRAY=30, we get 1% CPU utilization and each
large iteration takes 1 hour
4417
4418 int j_array[N_ELEMENTS_IN_J_ARRAY];
4419
4420 void
setup_datastructures_for_improve_learning_affine_prediction_model_leq_constra
int(int i) {
4421   int iterator; int j;
4422   nelements_of_constr = 0;
4423   ind[nelements_of_constr] = 0;
4424   val[nelements_of_constr] = -1.0;
4425   nelements_of_constr = nelements_of_constr + 1;
4426   for (iterator=0;iterator<N_ELEMENTS_IN_J_ARRAY;iterator++) {
4427     j = j_array[iterator];
4428     if
(isbitonein_bitposition_in_integerarray(manyruns[i].inputtoprogram,j)) {
4429       ind[nelements_of_constr] = iterator+1;
4430       val[nelements_of_constr] = 1.0;
4431       nelements_of_constr = nelements_of_constr + 1;
4432     }
4433   }
4434   ind[nelements_of_constr] = N_ELEMENTS_IN_J_ARRAY+1;
4435   val[nelements_of_constr] = 1.0;
4436   nelements_of_constr = nelements_of_constr + 1;
4437   // get_min_max_average_execution_time(&mint,&maxt,&averaget);
4438   // rhs = (manyruns[i].t-mint)/(maxt-mint) * 100.0;
4439   rhs = -manyruns[i].error;
4440 }
4441
4442 void
setup_datastructures_for_improve_learning_affine_prediction_model_leq_to_geq(
) {
4443   val[0] = val[0] * (-1.0);
4444 }
4445
4446 void write_w0_to_textfile_init_w0(char* fn_prefix) {
4447   char fn[200]; FILE* f; int j;
```

```
4448   double mint; double maxt; double averaget;
4449   get_min_max_average_execution_time(&mint,&maxt,&averaget);
4450   sprintf(fn,"%s_init_w0.txt",fn_prefix);
4451   f = fopen(fn, "w+" ); if (f==NULL) { printf("Error in
write_w0_to_textfile_init_w0\n"); fflush(stdout); exit(-1); }
4452   for (j=0;j<inputsizeinnumberofbits;j++) {
4453     fprintf(f,"%30.25lf\n", p_w0_value[j]);
4454   }
4455   fclose(f);
4456 }
4457 void write_b56_to_textfile_init_b56(char* fn_prefix) {
4458   char fn[200]; FILE* f;
4459   double mint; double maxt; double averaget;
4460   get_min_max_average_execution_time(&mint,&maxt,&averaget);
4461   sprintf(fn,"%s_init_b56.txt",fn_prefix);
4462   f = fopen(fn, "w+" ); if (f==NULL) { printf("Error in
write_b56_to_textfile_init_b56\n"); fflush(stdout); exit(-1); }
4463   fprintf(f,"%30.25lf\n", p_b56_value[0]);
4464   fclose(f);
4465 }
4466 void write_w0_and_b56_to_textfiles_z_w0_b56(char* fn_prefix) {
4467   write_w0_to_textfile_init_w0(fn_prefix);
4468   write_b56_to_textfile_init_b56(fn_prefix);
4469 }
4470
4471 void improve_learning_affine_prediction_model(int
n_examples_in_file,char* fn_prefix) {
4472   int large_iterator;
4473   int iterator;
4474   int j;
4475   int i;
4476   char gurobi_improve_learning_affine_prediction_model_log_file[200];
4477   char gurobi_improve_learning_affine_prediction_model_lp_file[200];
4478   char gurobi_improve_learning_affine_prediction_model_sol_file[200];
4479   fill_in_sorted_order_of_variables_in_affine_model();
4480   nelements_of_obj = N_ELEMENTS_IN_J_ARRAY+2; // z is 1st element, x_j
is 2nd element; ... x_j is 11th element; b56 is the 12th element
4481   dothememoryallocation_for_Gurobi_stuff();
4482   for
(large_iterator=0;large_iterator<N_LARGE_ITERATIONS;large_iterator++) {
4483     for (iterator=0;iterator<N_ELEMENTS_IN_J_ARRAY;iterator++) {
4484       j =
sorted_order_of_variables_in_affine_model[inputsizeinnumberofbits-1-
large_iterator*N_ELEMENTS_IN_J_ARRAY-iterator];
4485       j_array[iterator] = j;
4486     }
4487
sprintf(gurobi_improve_learning_affine_prediction_model_log_file,"%s_improve_
learning_affine_prediction_model_%d_%d.log",fn_prefix,large_iterator,j_array[
0]);
4488     sprintf(gurobi_improve_learning_affine_prediction_model_lp_file,
"%s_improve_learning_affine_prediction_model_%d_%d.lp",
fn_prefix,large_iterator,j_array[0]);
4489
sprintf(gurobi_improve_learning_affine_prediction_model_sol_file,"%s_improve_
learning_affine_prediction_model_%d_%d.sol",fn_prefix,large_iterator,j_array[
0]);
```

```
4490      error = GRBloadenv(&env,
gurobi_improve_learning_affine_prediction_model_log_file);
4491      if (error) goto QUIT;
4492
fillobj_for_objective_function_improve_learning_affine_prediction_model();
4493      error = GRBnewmodel(env, &model,
"improve_learning_affine_prediction_model", nelements_of_obj, obj, lb, ub,
vtype, NULL);
4494      if (error) goto QUIT;
4495      error = GRBsetintattr( model, GRB_INT_ATTR_MODELSENSE,
GRB_MINIMIZE);
4496      if (error) goto QUIT;
4497      error = GRBupdatemodel( model);
4498      if (error) goto QUIT;
4499      for (i=0;i<n_examples_in_file;i++) {
4500
setup_datastructures_for_improve_learning_affine_prediction_model_leq_constra
int(i);
4501        error = GRBaddconstr( model, nelements_of_constr, ind, val,
GRB_LESS_EQUAL, rhs, NULL );
4502        if (error) goto QUIT;
4503
setup_datastructures_for_improve_learning_affine_prediction_model_leq_to_geq(
);
4504        error = GRBaddconstr( model, nelements_of_constr, ind, val,
GRB_GREATER_EQUAL, rhs, NULL );
4505        if (error) goto QUIT;
4506      }
4507      error = GRBupdatemodel( model);
4508      if (error) goto QUIT;
4509      error = GRBoptimize(model);
4510      if (error) goto QUIT;
4511      error = GRBupdatemodel( model);
4512      if (error) goto QUIT;
4513      error = GRBgetintattr( model, GRB_INT_ATTR_STATUS, &optimstatus);
4514      if (error) goto QUIT;
4515      if (optimstatus == GRB_OPTIMAL) {
4516        error = GRBwrite( model,
gurobi_improve_learning_affine_prediction_model_sol_file);
4517        if (error) goto QUIT;
4518        error = GRBgetdblattr( model, GRB_DBL_ATTR_OBJVAL, &objval);
4519        if (error) goto QUIT;
4520        error = GRBgetdblattrarray( model, GRB_DBL_ATTR_X, 0,
nelements_of_obj, sol);
4521        if (error) goto QUIT;
4522        for (iterator=0;iterator<N_ELEMENTS_IN_J_ARRAY;iterator++) {
4523          j = j_array[iterator];
4524          p_w0_value[j]  = p_w0_value[j] + sol[iterator+1];
4525        }
4526        p_b56_value[0] = p_b56_value[0] + sol[N_ELEMENTS_IN_J_ARRAY+1];
4527        compute_error_and_prediction_based_on_w0_b56(n_examples_in_file);
4528        update_abs_error_index(n_examples_in_file);
4529      } else {
4530        printf("An error in improve_learning_affine_prediction_model. Did
not get an objective.\n"); fflush(stdout); exit(-1);
4531      }
4532      GRBfreemodel(model);
```

```
4533      GRBfreeenv(env);
4534    }
4535  QUIT:
4536    if (error) { printf("ERROR: %s\n", GRBgeterrormsg(env)); exit(1); }
4537    if (error) {
4538      GRBfreemodel(model);
4539      GRBfreeenv(env);
4540    }
4541    freethememory_for_Gurobi_stuff();
4542    printf("Almost finish improve_learning_affine_prediction_model\n");
fflush(stdout);
4543    write_w0_and_b56_to_textfiles_z_w0_b56(fn_prefix);
4544    printf("Finish improve_learning_affine_prediction_model\n");
fflush(stdout);
4545  }
4546
4547  // with n_examples_selected = 1000000
4548  //   #define NBASISFUNCTIONS 100000     // it takes 1 hour to run the
entire program if NBASISFUNCTIONS=100000
4549  //   #define NBASISFUNCTIONS 500000     // it is expected to take 12
hours to run the entire program if NBASISFUNCTIONS=500000
4550  //   #define NBASISFUNCTIONS 1000000    // it is expected to take 48
hours to run the entire program if NBASISFUNCTIONS=1000000
4551  //   #define NBASISFUNCTIONS 2000000    // it takes 37 hours to run the
entire program if NBASISFUNCTIONS=2000000
4552  //   #define NBASISFUNCTIONS 4000000    // it takes 3 days to run the
entire program if NBASISFUNCTIONS=4000000
4553  //   #define NBASISFUNCTIONS 8000000    // it takes 6 days to run the
entire program if NBASISFUNCTIONS=8000000
4554  // with n_examples_selected = 2000000
4555  // #define NBASISFUNCTIONS 8000000    // it is expected to take 12 days
to run the entire program if NBASISFUNCTIONS=8000000
4556  // #define NBASISFUNCTIONS 1000000
4557  // #define NBASISFUNCTIONS 100
4558  // #define NBASISFUNCTIONS 1000
4559  // #define NBASISFUNCTIONS 10
4560  // #define NBASISFUNCTIONS 100
4561  // #define NBASISFUNCTIONS 1000
4562  // #define NBASISFUNCTIONS 10000
4563  // #define NBASISFUNCTIONS 100000
4564  // #define NBASISFUNCTIONS 1000000
4565  // #define NBASISFUNCTIONS 1000
4566  #define NBASISFUNCTIONS 10000
4567
4568  #define SELECT_BESTBASISFUNCTION_AMONG 100
4569  #define NEXAMPLES_USED_FOR_SELECTING_BASISFUNCTIONS 2000000
4570
4571  struct basisfunction {
4572    int bit1_index; // this is a value in the range [0,nbits-1]
4573    int bit2_index; // this is a value in the range [0,nbits-1] s.t.
bit1_index < bit2_index
4574    int bit1_value; // this is a value in {0,1}
4575    int bit2_value; // this is a value in {0,1}
4576    double coefficient;
4577    // int score;
4578    double score;
4579    double delta;
```

```
4580 };
4581 struct basisfunction basisfunctions[NBASISFUNCTIONS];
4582
4583 struct basisfunction generate_new_basisfunction() {
4584   int temp1; int temp2;
4585   struct basisfunction ret;
4586   temp1 = random()%(inputsizeinnumberofbits-1);
4587   temp2 = random()%(inputsizeinnumberofbits-2);
4588   if (temp2>=temp1) {
4589     temp2 = temp2 + 1;
4590   }
4591   if (temp1>temp2) {
4592     swapint(&temp1,&temp2);
4593   }
4594   ret.bit1_index = temp1;
4595   ret.bit2_index = temp2;
4596   ret.bit1_value = random()%2;
4597   ret.bit2_value = random()%2;
4598   return ret;
4599 }
4600
4601 int basisfunctions_are_identical_quad(struct basisfunction
basisfunction1,struct basisfunction basisfunction2) {
4602   if (basisfunction1.bit1_index==basisfunction2.bit1_index) {
4603     if (basisfunction1.bit2_index==basisfunction2.bit2_index) {
4604       if (basisfunction1.bit1_value==basisfunction2.bit1_value) {
4605         if (basisfunction1.bit2_value==basisfunction2.bit2_value) {
4606           return 1;
4607         } else {
4608           return 0;
4609         }
4610       } else {
4611         return 0;
4612       }
4613     } else {
4614       return 0;
4615     }
4616   } else {
4617     return 0;
4618   }
4619 }
4620
4621 int is_basisfunction_in(int n_elements_already_in_array,struct
basisfunction new_basisfunction) {
4622   int iterator;
4623   for (iterator=0;iterator<n_elements_already_in_array;iterator++) {
4624     if
(basisfunctions_are_identical_quad(basisfunctions[iterator],new_basisfunction
)) {
4625       return 1;
4626     }
4627   }
4628   return 0;
4629 }
4630
4631 int evaluate_basis_on_value_of_two_bits_given(struct basisfunction*
p_basisfunction,int vb1,int vb2) {
```

```
4632   if ((p_basisfunction->bit1_value==vb1) && (p_basisfunction-
>bit2_value==vb2)) {
4633     return 1;
4634   } else {
4635     return 0;
4636   }
4637 }
4638
4639 int evaluate_basis_on_programinput(struct basisfunction*
p_basisfunction,int* p_anintarray) {
4640   return evaluate_basis_on_value_of_two_bits_given(p_basisfunction,
4641
isbitonein_bitposition_in_integerarray(p_anintarray,p_basisfunction-
>bit1_index),
4642
isbitonein_bitposition_in_integerarray(p_anintarray,p_basisfunction-
>bit2_index)
4643                                                         );
4644 }
4645
4646 int evaluate_basis_on_example(struct basisfunction* p_basisfunction,int
i) {
4647   return
evaluate_basis_on_programinput(p_basisfunction,manyruns[i].inputtoprogram);
4648 }
4649
4650 void compute_coefficient(struct basisfunction* p_basisfunction,int
n_examples_in_file) {
4651   int iterator; int index_to_large_abs_error; int i; int bit_value;
4652   double sum0; int count0;
4653   double sum1; int count1;
4654   sum0 = 0.0; count0 = 0;
4655   sum1 = 0.0; count1 = 0;
4656   for
(iterator=0;iterator<NEXAMPLES_USED_FOR_SELECTING_BASISFUNCTIONS;iterator++)
{
4657     index_to_large_abs_error = n_examples_in_file-1-iterator;
4658     i = abs_error_index_of_examples[index_to_large_abs_error];
4659     bit_value = evaluate_basis_on_example(p_basisfunction,i);
4660     if (bit_value==0) {
4661       sum0 = sum0 + manyruns[i].error; count0 = count0 + 1;
4662     } else {
4663       if (bit_value==1) {
4664         sum1 = sum1 + manyruns[i].error; count1 = count1 + 1;
4665       } else {
4666         printf("Error in compute_coefficient\n"); exit(-1);
4667       }
4668     }
4669   }
4670   if ((count1>0) && (count0>0)) {
4671     p_basisfunction->coefficient = -((sum1/count1) - (sum0/count0));
4672   } else {
4673     printf("In compute_coefficient. This is odd but not necessarily an
error\n");
4674     p_basisfunction->coefficient = 0.0;
4675   }
4676 }
```

```
4677
4678 struct myargstruct4 {
4679   int loindex;
4680   int hiindex;
4681   int n_examples_in_file;
4682 };
4683 pthread_t threads4[NUM_THREADS_USED_FOR_WORK];
4684 pthread_attr_t attrs4[NUM_THREADS_USED_FOR_WORK];
4685 struct myargstruct4 myargstruct_array4[NUM_THREADS_USED_FOR_WORK];
4686
4687 void set_indices_in_myargstruct_array4(int th,int n_examples_in_file) {
4688   int objects_per_worker;
4689   if (NBASISFUNCTIONS%NUM_THREADS_USED_FOR_WORK==0) {
4690     objects_per_worker = NBASISFUNCTIONS/NUM_THREADS_USED_FOR_WORK;
4691   } else {
4692     objects_per_worker = NBASISFUNCTIONS/NUM_THREADS_USED_FOR_WORK + 1;
4693   }
4694   myargstruct_array4[th].loindex = th    * objects_per_worker;
4695   myargstruct_array4[th].hiindex = (th+1)* objects_per_worker - 1;
4696   if (myargstruct_array4[th].hiindex>NBASISFUNCTIONS-1) {
4697     myargstruct_array4[th].hiindex = NBASISFUNCTIONS-1;
4698   }
4699   myargstruct_array4[th].n_examples_in_file = n_examples_in_file;
4700 }
4701
4702 __attribute__((optimize("-O3"))) void* worker4(void *p) {
4703   int iterator;
4704   for (iterator=((struct myargstruct4*) p)->loindex;iterator<=((struct
myargstruct4*) p)->hiindex;iterator++) {
4705     compute_coefficient(&(basisfunctions[iterator]),((struct
myargstruct4*) p)->n_examples_in_file);
4706   }
4707   pthread_exit(NULL);
4708 }
4709
4710 void compute_coefficients(int n_examples_in_file) {
4711   int rc; int th; void* status;
4712   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
4713      pthread_attr_init(&(attrs4[th]));
4714      pthread_attr_setdetachstate(&(attrs4[th]),PTHREAD_CREATE_JOINABLE);
4715      set_indices_in_myargstruct_array4(th,n_examples_in_file);
4716      rc = pthread_create(&(threads4[th]),&(attrs4[th]),worker4,(void*)
(&(myargstruct_array4[th])));
4717      if (rc){
4718         printf("ERROR; return code from pthread_create() is %d\n", rc);
4719         exit(-1);
4720      }
4721   }
4722   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
4723      rc = pthread_join(threads4[th], &status);
4724      if (rc) {
4725         printf("ERROR; return code from pthread_join() is %d\n", rc);
4726         exit(-1);
4727      }
4728   }
4729   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
4730     pthread_attr_destroy( &(attrs4[th]) );
```

```
4731   }
4732 }
4733
4734 int cmpfunc_sort_basisfunctions(const void * a, const void * b) {
4735   struct basisfunction* p_a;
4736   struct basisfunction* p_b;
4737   double abs_coefficient_a;
4738   double abs_coefficient_b;
4739   p_a = (struct basisfunction*)a;
4740   p_b = (struct basisfunction*)b;
4741   abs_coefficient_a = fabs(p_a->coefficient);
4742   abs_coefficient_b = fabs(p_b->coefficient);
4743   if (abs_coefficient_a < abs_coefficient_b) {
4744     return -1;
4745   } else {
4746     if (abs_coefficient_a == abs_coefficient_b) {
4747       return 0;
4748     } else {
4749       return 1;
4750     }
4751   }
4752 }
4753
4754 void sort_basis_functions() {
4755   qsort(basisfunctions, NBASISFUNCTIONS, sizeof(struct basisfunction),
cmpfunc_sort_basisfunctions);
4756 }
4757
4758 void print_basis_functions_to_file(char* fn_prefix) {
4759   char fn[200]; FILE* f; int iterator;
4760   sprintf(fn,"%s_basis_functions",fn_prefix);
4761   f = fopen(fn, "w+");
4762   if (f==NULL) { printf("In print_basis_functions_to_file. Opening file.
failed\n"); exit(-1); }
4763   for (iterator=0;iterator<NBASISFUNCTIONS;iterator++) {
4764     // fprintf(f,"%d %d %d %d %18.15lf\n",
basisfunctions[iterator].bit1_index,basisfunctions[iterator].bit2_index,basis
functions[iterator].bit1_value,basisfunctions[iterator].bit2_value,
4765     //  basisfunctions[iterator].coefficient);
4766     // fprintf(f,"%d %d %d %d %18.15lf %d\n",
basisfunctions[iterator].bit1_index,basisfunctions[iterator].bit2_index,basis
functions[iterator].bit1_value,basisfunctions[iterator].bit2_value,
4767     //  basisfunctions[iterator].coefficient,
basisfunctions[iterator].score);
4768     fprintf(f,"%d %d %d %d %18.15lf %18.15lf %18.15lf\n",
basisfunctions[iterator].bit1_index,basisfunctions[iterator].bit2_index,basis
functions[iterator].bit1_value,basisfunctions[iterator].bit2_value,
4769       basisfunctions[iterator].coefficient,
basisfunctions[iterator].score, basisfunctions[iterator].delta);
4770   }
4771   fclose( f);
4772 }
4773
4774 void read_basis_functions_from_file(char* fn_prefix) {
4775   char fn[200]; FILE* f; int iterator; int n_obtained;
4776   sprintf(fn,"%s_basis_functions",fn_prefix);
4777   f = fopen(fn, "r");
```

```
4778   if (f==NULL) { printf("In read_basis_functions_from_file. Opening
file. failed\n"); exit(-1); }
4779   for (iterator=0;iterator<NBASISFUNCTIONS;iterator++) {
4780     // n_obtained = fscanf(f,"%d %d %d %d %lf",
4781     //   &(basisfunctions[iterator].bit1_index),
4782     //   &(basisfunctions[iterator].bit2_index),
4783     //   &(basisfunctions[iterator].bit1_value),
4784     //   &(basisfunctions[iterator].bit2_value),
4785     //   &(basisfunctions[iterator].coefficient)
4786     //                     );
4787     // n_obtained = fscanf(f,"%d %d %d %d %lf %d",
4788     //   &(basisfunctions[iterator].bit1_index),
4789     //   &(basisfunctions[iterator].bit2_index),
4790     //   &(basisfunctions[iterator].bit1_value),
4791     //   &(basisfunctions[iterator].bit2_value),
4792     //   &(basisfunctions[iterator].coefficient),
4793     //   &(basisfunctions[iterator].score)
4794     //                     );
4795     n_obtained = fscanf(f,"%d %d %d %d %lf %lf %lf",
4796       &(basisfunctions[iterator].bit1_index),
4797       &(basisfunctions[iterator].bit2_index),
4798       &(basisfunctions[iterator].bit1_value),
4799       &(basisfunctions[iterator].bit2_value),
4800       &(basisfunctions[iterator].coefficient),
4801       &(basisfunctions[iterator].score),
4802       &(basisfunctions[iterator].delta)
4803                         );
4804     // if (n_obtained<5) {
4805     // if (n_obtained<6) {
4806     if (n_obtained<7) {
4807       printf("Error in read_basis_functions_from_file. Number of
elements obtained = %d.\n",n_obtained); exit(-1);
4808     }
4809   }
4810   fclose( f);
4811 }
4812
4813 void set_all_coefficients_of_basis_functions_to_zero() {
4814   int iterator;
4815   for (iterator=0;iterator<NBASISFUNCTIONS;iterator++) {
4816     basisfunctions[iterator].coefficient = 0.0;
4817   }
4818 }
4819
4820 void save_all_coefficients_of_basis_functions() {
4821   print_basis_functions_to_file("temp");
4822 }
4823
4824 void restore_all_coefficients_of_basis_functions() {
4825   read_basis_functions_from_file("temp");
4826 }
4827
4828 void find_suitable_basis_functions(char* fn_prefix,int
n_examples_in_file) {
4829   int iterator; struct basisfunction new_basisfunction;
4830   for (iterator=0;iterator<NBASISFUNCTIONS;iterator++) {
4831     new_basisfunction = generate_new_basisfunction();
```

```
4832      while (is_basisfunction_in(iterator,new_basisfunction)) {
4833        new_basisfunction = generate_new_basisfunction();
4834      }
4835      basisfunctions[iterator] = new_basisfunction;
4836    }
4837    compute_coefficients(n_examples_in_file);
4838    sort_basis_functions();
4839    print_basis_functions_to_file(fn_prefix);
4840  }
4841
4842  void find_WCET_input_from_prediction_model_w0_b56(char*
fn_prefix_string,int* p_int_array,double* p_t) {
4843    int j;
4844    clear_int_array(p_int_array);
4845    for (j=0;j<inputsizeinnumberofbits;j++) {
4846      if (p_w0_value[j]>0.0) {
4847        setbitonein_bitposition_in_integerarray(p_int_array,j);
4848      }
4849    }
4850    *p_t = p_b56_value[0];
4851    for (j=0;j<inputsizeinnumberofbits;j++) {
4852      if (isbitonein_bitposition_in_integerarray(p_int_array,j)) {
4853        *p_t = *p_t + p_w0_value[j];
4854      }
4855    }
4856  }
4857
4858  struct myargstruct5 {
4859    struct basisfunction* p_basisfunction;
4860    int* p_int_array;
4861    int loindex;
4862    int hiindex;
4863    double partial_sum;
4864  };
4865  pthread_t threads5[NUM_THREADS_USED_FOR_WORK];
4866  pthread_attr_t attrs5[NUM_THREADS_USED_FOR_WORK];
4867  struct myargstruct5 myargstruct_array5[NUM_THREADS_USED_FOR_WORK];
4868
4869  void set_indices_in_myargstruct_array5(int th,struct basisfunction*
new_p_basisfunction,int* new_p_int_array) {
4870    int objects_per_worker;
4871    myargstruct_array5[th].p_basisfunction = new_p_basisfunction;
4872    myargstruct_array5[th].p_int_array = new_p_int_array;
4873    if (NBASISFUNCTIONS%NUM_THREADS_USED_FOR_WORK==0) {
4874      objects_per_worker = NBASISFUNCTIONS/NUM_THREADS_USED_FOR_WORK;
4875    } else {
4876      objects_per_worker = NBASISFUNCTIONS/NUM_THREADS_USED_FOR_WORK + 1;
4877    }
4878    myargstruct_array5[th].loindex = th    * objects_per_worker;
4879    myargstruct_array5[th].hiindex = (th+1)* objects_per_worker - 1;
4880    if (myargstruct_array5[th].hiindex>NBASISFUNCTIONS-1) {
4881      myargstruct_array5[th].hiindex = NBASISFUNCTIONS-1;
4882    }
4883    myargstruct_array5[th].partial_sum = 0.0;
4884  }
4885
4886  __attribute__((optimize("-O3"))) void* worker5(void *p) {
```

```
4887   int* p_int_array;
4888   double partial_sum; int iterator;
4889   partial_sum = 0.0;
4890   for (iterator=((struct myargstruct5*) p)->loindex;iterator<=((struct
myargstruct5*) p)->hiindex;iterator++) {
4891     p_int_array = ((struct myargstruct5*) p)->p_int_array;
4892     if
(evaluate_basis_on_programinput(&(basisfunctions[iterator]),p_int_array)) {
4893       if (
4894            (basisfunctions[iterator].bit1_index==((((struct
myargstruct5*) p)->p_basisfunction)->bit1_index)) ||
4895            (basisfunctions[iterator].bit2_index==((((struct
myargstruct5*) p)->p_basisfunction)->bit1_index)) ||
4896            (basisfunctions[iterator].bit1_index==((((struct
myargstruct5*) p)->p_basisfunction)->bit2_index)) ||
4897            (basisfunctions[iterator].bit2_index==((((struct
myargstruct5*) p)->p_basisfunction)->bit2_index))
4898          ) {
4899           partial_sum = partial_sum +
basisfunctions[iterator].coefficient;
4900       }
4901     }
4902   }
4903   ((struct myargstruct5*) p)->partial_sum = partial_sum;
4904   pthread_exit(NULL);
4905 }
4906
4907 double calc_contrib(struct basisfunction* p_basisfunction,int*
p_int_array) {
4908   double sum;
4909   int rc; int th; void* status;
4910   sum = p_w0_value[p_basisfunction->bit1_index] *
isbitonein_bitposition_in_integerarray(p_int_array,p_basisfunction-
>bit1_index) +
4911         p_w0_value[p_basisfunction->bit2_index] *
isbitonein_bitposition_in_integerarray(p_int_array,p_basisfunction-
>bit2_index);
4912   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
4913      pthread_attr_init(&(attrs5[th]));
4914      pthread_attr_setdetachstate(&(attrs5[th]),PTHREAD_CREATE_JOINABLE);
4915      set_indices_in_myargstruct_array5(th,p_basisfunction,p_int_array);
4916      rc = pthread_create(&(threads5[th]),&(attrs5[th]),worker5,(void*)
(&(myargstruct_array5[th])));
4917      if (rc){
4918         printf("ERROR; return code from pthread_create() is %d\n", rc);
4919         exit(-1);
4920      }
4921   }
4922   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
4923      rc = pthread_join(threads5[th], &status);
4924      if (rc) {
4925         printf("ERROR; return code from pthread_join() is %d\n", rc);
4926         exit(-1);
4927      }
4928   }
4929   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
4930     pthread_attr_destroy( &(attrs5[th]) );
```

```
4931   }
4932   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
4933     sum = sum + myargstruct_array5[th].partial_sum;
4934   }
4935   return sum;
4936 }
4937
4938 int evaluate_basis_on_value_of_two_bits_given_v2(struct basisfunction*
p_basisfunction,int vb1,int vb2) {
4939   if ((p_basisfunction->bit1_value==vb1) && (p_basisfunction-
>bit2_value==vb2)) {
4940     return 1;
4941   } else {
4942     return 0;
4943   }
4944 }
4945
4946 int evaluate_basis_on_inputtoprogram_v2(struct basisfunction*
p_basisfunction,int* p_anintarray) {
4947   return evaluate_basis_on_value_of_two_bits_given_v2(p_basisfunction,
4948
isbitonein_bitposition_in_integerarray(p_anintarray,p_basisfunction-
>bit1_index),
4949
isbitonein_bitposition_in_integerarray(p_anintarray,p_basisfunction-
>bit2_index)
4950                                                       );
4951 }
4952
4953 int evaluate_basis_on_example_v2_seq(struct basisfunction*
p_basisfunction,int i) {
4954   return
evaluate_basis_on_inputtoprogram_v2(p_basisfunction,manyruns[i].inputtoprogra
m);
4955 }
4956
4957 // int evaluate_basis_on_example_v2_par(struct basisfunction*
p_basisfunction,int i) {
4958 //   return
evaluate_basis_on_inputtoprogram_v2(p_basisfunction,manyruns[i].inputtoprogra
m);
4959 // }
4960
4961 double
compute_prediction_on_inputtoprogram_w0_b56_basis_functions_seq(int*
p_anintarray) {
4962   double sum; int j; int basisfunctionindex;
4963   sum = p_b56_value[0];
4964   for (j=0;j<inputsizeinnumberofbits;j++) {
4965     if (isbitonein_bitposition_in_integerarray(p_anintarray,j)) {
4966       sum = sum + p_w0_value[j];
4967     }
4968   }
4969   for
(basisfunctionindex=0;basisfunctionindex<NBASISFUNCTIONS;basisfunctionindex++
) {
```

```
4970      if
(evaluate_basis_on_inputtoprogram_v2(&(basisfunctions[basisfunctionindex]),p_
anintarray)) {
4971        sum = sum + basisfunctions[basisfunctionindex].coefficient;
4972      }
4973    }
4974    return sum;
4975  }
4976  double compute_prediction_on_example_w0_b56_basis_functions_seq( int i)
{
4977    return
compute_prediction_on_inputtoprogram_w0_b56_basis_functions_seq(manyruns[i].i
nputtoprogram);
4978  }
4979
4980  struct myargstruct5b {
4981    int* p_int_array;
4982    int loindex;
4983    int hiindex;
4984    double sum;
4985  };
4986  pthread_t threads5b[NUM_THREADS_USED_FOR_WORK];
4987  pthread_attr_t attrs5b[NUM_THREADS_USED_FOR_WORK];
4988  struct myargstruct5b myargstruct_array5b[NUM_THREADS_USED_FOR_WORK];
4989
4990  void set_indices_in_myargstruct_array5b(int th,int* new_p_int_array) {
4991    int objects_per_worker;
4992    myargstruct_array5b[th].p_int_array = new_p_int_array;
4993    if (NBASISFUNCTIONS%NUM_THREADS_USED_FOR_WORK==0) {
4994      objects_per_worker = NBASISFUNCTIONS/NUM_THREADS_USED_FOR_WORK;
4995    } else {
4996      objects_per_worker = NBASISFUNCTIONS/NUM_THREADS_USED_FOR_WORK + 1;
4997    }
4998    myargstruct_array5b[th].loindex = th    * objects_per_worker;
4999    myargstruct_array5b[th].hiindex = (th+1)* objects_per_worker - 1;
5000    if (myargstruct_array5b[th].hiindex>NBASISFUNCTIONS-1) {
5001      myargstruct_array5b[th].hiindex = NBASISFUNCTIONS-1;
5002    }
5003    myargstruct_array5b[th].sum = 0.0;
5004  }
5005
5006  __attribute__((optimize("-O3"))) void* worker5b(void *p) {
5007    int* p_int_array;
5008    double sum;
5009    // int iterator;
5010    int basisfunctionindex;
5011    sum = 0.0;
5012    // for (iterator=((struct myargstruct5b*) p)-
>loindex;iterator<=((struct myargstruct5b*) p)->hiindex;iterator++) {
5013    for (basisfunctionindex=((struct myargstruct5b*) p)-
>loindex;basisfunctionindex<=((struct myargstruct5b*) p)-
>hiindex;basisfunctionindex++) {
5014      p_int_array = ((struct myargstruct5b*) p)->p_int_array;
5015      // if
(evaluate_basis_on_programinput(&(basisfunctions[iterator]),p_int_array)) {
5016      //  sum = sum + basisfunctions[iterator].coefficient;
5017      // }
```

```
5018     if
(evaluate_basis_on_inputtoprogram_v2(&(basisfunctions[basisfunctionindex]),p_
int_array)) {
5019       sum = sum + basisfunctions[basisfunctionindex].coefficient;
5020     }
5021   }
5022   ((struct myargstruct5b*) p)->sum = sum;
5023   pthread_exit(NULL);
5024 }
5025
5026 double compute_sum_of_basis_functions(int* p_anintarray) {
5027   int rc; int th; void* status; double sum;
5028   sum = 0.0;
5029   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
5030      pthread_attr_init(&(attrs5b[th]));
5031
pthread_attr_setdetachstate(&(attrs5b[th]),PTHREAD_CREATE_JOINABLE);
5032      // set_indices_in_myargstruct_array5b(th,p_int_array);
5033      set_indices_in_myargstruct_array5b(th,p_anintarray);
5034      rc =
pthread_create(&(threads5b[th]),&(attrs5b[th]),worker5b,(void*)
(&(myargstruct_array5b[th])));
5035      if (rc){
5036         printf("ERROR; return code from pthread_create() is %d\n", rc);
5037         exit(-1);
5038      }
5039   }
5040   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
5041      rc = pthread_join(threads5b[th], &status);
5042      if (rc) {
5043         printf("ERROR; return code from pthread_join() is %d\n", rc);
5044         exit(-1);
5045      }
5046   }
5047   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
5048     pthread_attr_destroy( &(attrs5b[th]) );
5049   }
5050   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
5051     sum = sum + myargstruct_array5b[th].sum;
5052   }
5053   return sum;
5054 }
5055
5056 double
compute_prediction_on_inputtoprogram_w0_b56_basis_functions_par(int*
p_anintarray) {
5057   double sum; int j; int basisfunctionindex;
5058   sum = p_b56_value[0];
5059   for (j=0;j<inputsizeinnumberofbits;j++) {
5060     if (isbitonein_bitposition_in_integerarray(p_anintarray,j)) {
5061       sum = sum + p_w0_value[j];
5062     }
5063   }
5064   sum = sum + compute_sum_of_basis_functions(p_anintarray);
5065   return sum;
5066 }
5067
```

```
5068 // double compute_prediction_on_example_w0_b56_basis_functions_seq( int
i) {
5069 //  return
compute_prediction_on_inputtoprogram_w0_b56_basis_functions_seq(manyruns[i].i
nputtoprogram);
5070 // }
5071 // double compute_prediction_on_example_w0_b56_basis_functions_par( int
i) {
5072 //  return
compute_prediction_on_inputtoprogram_w0_b56_basis_functions_par(manyruns[i].i
nputtoprogram);
5073 // }
5074
5075 void potentially_modify_bits_considering_basis_function(struct
basisfunction* p_basisfunction,int* p_int_array) {
5076   double t00;
5077   double t01;
5078   double t10;
5079   double t11;
5080   setbitzeroin_bitposition_in_integerarray( p_int_array,p_basisfunction-
>bit1_index);
5081   setbitzeroin_bitposition_in_integerarray( p_int_array,p_basisfunction-
>bit2_index);
5082   t00 =
compute_prediction_on_inputtoprogram_w0_b56_basis_functions_par(p_int_array);
5083   setbitzeroin_bitposition_in_integerarray( p_int_array,p_basisfunction-
>bit1_index);
5084   setbitonein_bitposition_in_integerarray(  p_int_array,p_basisfunction-
>bit2_index);
5085   t01 =
compute_prediction_on_inputtoprogram_w0_b56_basis_functions_par(p_int_array);
5086   setbitonein_bitposition_in_integerarray(  p_int_array,p_basisfunction-
>bit1_index);
5087   setbitzeroin_bitposition_in_integerarray( p_int_array,p_basisfunction-
>bit2_index);
5088   t10 =
compute_prediction_on_inputtoprogram_w0_b56_basis_functions_par(p_int_array);
5089   setbitonein_bitposition_in_integerarray(  p_int_array,p_basisfunction-
>bit1_index);
5090   setbitonein_bitposition_in_integerarray(  p_int_array,p_basisfunction-
>bit2_index);
5091   t11 =
compute_prediction_on_inputtoprogram_w0_b56_basis_functions_par(p_int_array);
5092   if (t00>t01) {
5093     if (t10>t11) {
5094       if (t00>t10) {
5095
setbitzeroin_bitposition_in_integerarray(p_int_array,p_basisfunction-
>bit1_index);
5096
setbitzeroin_bitposition_in_integerarray(p_int_array,p_basisfunction-
>bit2_index);
5097       } else {
5098         setbitonein_bitposition_in_integerarray(
p_int_array,p_basisfunction->bit1_index);
```

```
5099
setbitzeroin_bitposition_in_integerarray(p_int_array,p_basisfunction-
>bit2_index);
5100       }
5101     } else {
5102       if (t00>t11) {
5103
setbitzeroin_bitposition_in_integerarray(p_int_array,p_basisfunction-
>bit1_index);
5104
setbitzeroin_bitposition_in_integerarray(p_int_array,p_basisfunction-
>bit2_index);
5105       } else {
5106         setbitonein_bitposition_in_integerarray(
p_int_array,p_basisfunction->bit1_index);
5107         setbitonein_bitposition_in_integerarray(
p_int_array,p_basisfunction->bit2_index);
5108       }
5109     }
5110   } else {
5111     if (t10>t11) {
5112       if (t01>t10) {
5113
setbitzeroin_bitposition_in_integerarray(p_int_array,p_basisfunction-
>bit1_index);
5114         setbitonein_bitposition_in_integerarray(
p_int_array,p_basisfunction->bit2_index);
5115       } else {
5116         setbitonein_bitposition_in_integerarray(
p_int_array,p_basisfunction->bit1_index);
5117
setbitzeroin_bitposition_in_integerarray(p_int_array,p_basisfunction-
>bit2_index);
5118       }
5119     } else {
5120       if (t01>t11) {
5121
setbitzeroin_bitposition_in_integerarray(p_int_array,p_basisfunction-
>bit1_index);
5122         setbitonein_bitposition_in_integerarray(
p_int_array,p_basisfunction->bit2_index);
5123       } else {
5124         setbitonein_bitposition_in_integerarray(
p_int_array,p_basisfunction->bit1_index);
5125         setbitonein_bitposition_in_integerarray(
p_int_array,p_basisfunction->bit2_index);
5126       }
5127     }
5128   }
5129 }
5130
5131 void find_WCET_input_from_prediction_model_w0_b56_w1_b1_w5(char*
fn_prefix_string,int* p_int_array,double* p_t) {
5132   int j; int iterator; int b;
5133
5134   clear_int_array(p_int_array);
5135   for (j=0;j<inputsizeinnumberofbits;j++) {
```

```
5136      if (p_w0_value[j]>0.0) {
5137        setbitonein_bitposition_in_integerarray(p_int_array,j);
5138      }
5139    }
5140
5141    *p_t = p_b56_value[0];
5142    for (j=0;j<inputsizeinnumberofbits;j++) {
5143      if (isbitonein_bitposition_in_integerarray(p_int_array,j)) {
5144        *p_t = *p_t + p_w0_value[j];
5145      }
5146    }
5147    for (iterator=0;iterator<NBASISFUNCTIONS;iterator++) {
5148      b =
evaluate_basis_on_programinput(&(basisfunctions[iterator]),p_int_array);
5149      if (b) {
5150        *p_t = *p_t + basisfunctions[iterator].coefficient;
5151      }
5152    }
5153    printf("Execution-time prediction before modifying two-bit basis
functions = %lf\n",*p_t);
5154
5155    for (iterator=0;iterator<NBASISFUNCTIONS;iterator++) {
5156
potentially_modify_bits_considering_basis_function(&(basisfunctions[iterator]
),p_int_array);
5157    }
5158
5159    *p_t = p_b56_value[0];
5160    for (j=0;j<inputsizeinnumberofbits;j++) {
5161      if (isbitonein_bitposition_in_integerarray(p_int_array,j)) {
5162        *p_t = *p_t + p_w0_value[j];
5163      }
5164    }
5165    for (iterator=0;iterator<NBASISFUNCTIONS;iterator++) {
5166      b =
evaluate_basis_on_programinput(&(basisfunctions[iterator]),p_int_array);
5167      if (b) {
5168        *p_t = *p_t + basisfunctions[iterator].coefficient;
5169      }
5170    }
5171    printf("Execution-time prediction after modifying two-bit basis
functions = %lf\n",*p_t);
5172 }
5173
5174 #define ADJUST_W0_B56_NEPOCHS 1
5175 #define ADJUST_W0_LEARNINGRATE 1.00
5176
5177 struct myargstruct6 {
5178    int j;
5179    int loindex;
5180    int hiindex;
5181    double minerr0; double maxerr0; double minerr1; double maxerr1;
5182 };
5183 pthread_t threads6[NUM_THREADS_USED_FOR_WORK];
5184 pthread_attr_t attrs6[NUM_THREADS_USED_FOR_WORK];
5185 struct myargstruct6 myargstruct_array6[NUM_THREADS_USED_FOR_WORK];
5186
```

```
5187 void set_indices_in_myargstruct_array6(int th,int n_examples_in_file,int
j) {
5188   int objects_per_worker;
5189   myargstruct_array6[th].j = j;
5190   if (n_examples_in_file%NUM_THREADS_USED_FOR_WORK==0) {
5191     objects_per_worker = n_examples_in_file/NUM_THREADS_USED_FOR_WORK;
5192   } else {
5193     objects_per_worker = n_examples_in_file/NUM_THREADS_USED_FOR_WORK +
1;
5194   }
5195   myargstruct_array6[th].loindex = th    * objects_per_worker;
5196   myargstruct_array6[th].hiindex = (th+1)* objects_per_worker - 1;
5197   if (myargstruct_array6[th].hiindex>n_examples_in_file-1) {
5198     myargstruct_array6[th].hiindex = n_examples_in_file-1;
5199   }
5200 }
5201
5202 __attribute__((optimize("-O3"))) void* worker6(void *p) {
5203   int i;
5204   int bit_equals_0_result_available; int bit_equals_1_result_available;
5205   bit_equals_0_result_available = 0;
5206   bit_equals_1_result_available = 0;
5207   for (i=((struct myargstruct6*) p)->loindex;i<=((struct myargstruct6*)
p)->hiindex;i++) {
5208     if
(isbitonein_bitposition_in_integerarray(manyruns[i].inputtoprogram,((struct
myargstruct6*) p)->j)) {
5209       if (bit_equals_1_result_available) {
5210         if (manyruns[i].error<((struct myargstruct6*) p)->minerr1) {
((struct myargstruct6*) p)->minerr1 = manyruns[i].error; }
5211         if (manyruns[i].error>((struct myargstruct6*) p)->maxerr1) {
((struct myargstruct6*) p)->maxerr1 = manyruns[i].error; }
5212       } else {
5213         ((struct myargstruct6*) p)->minerr1 = manyruns[i].error;
5214         ((struct myargstruct6*) p)->maxerr1 = manyruns[i].error;
5215         bit_equals_1_result_available = 1;
5216       }
5217     } else {
5218       if (bit_equals_0_result_available) {
5219         if (manyruns[i].error<((struct myargstruct6*) p)->minerr0) {
((struct myargstruct6*) p)->minerr0 = manyruns[i].error; }
5220         if (manyruns[i].error>((struct myargstruct6*) p)->maxerr0) {
((struct myargstruct6*) p)->maxerr0 = manyruns[i].error; }
5221       } else {
5222         ((struct myargstruct6*) p)->minerr0 = manyruns[i].error;
5223         ((struct myargstruct6*) p)->maxerr0 = manyruns[i].error;
5224         bit_equals_0_result_available = 1;
5225       }
5226     }
5227   }
5228   if (!bit_equals_0_result_available) {
5229     ((struct myargstruct6*) p)->minerr0 = 0.0;
5230     ((struct myargstruct6*) p)->maxerr0 = 0.0;
5231   }
5232   if (!bit_equals_1_result_available) {
5233     ((struct myargstruct6*) p)->minerr1 = 0.0;
5234     ((struct myargstruct6*) p)->maxerr1 = 0.0;
```

```
5235   }
5236   pthread_exit(NULL);
5237 }
5238
5239 void compute_minerr_maxerr_for_bit_for_0_and_1(int
n_examples_in_file,int j,double* p_minerr0,double* p_maxerr0,double*
p_minerr1,double* p_maxerr1) {
5240   int rc; int th; void* status;
5241   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
5242     pthread_attr_init(&(attrs6[th]));
5243     pthread_attr_setdetachstate(&(attrs6[th]),PTHREAD_CREATE_JOINABLE);
5244     set_indices_in_myargstruct_array6(th,n_examples_in_file,j);
5245     rc = pthread_create(&(threads6[th]),&(attrs6[th]),worker6,(void*)
(&(myargstruct_array6[th])));
5246     if (rc){
5247       printf("ERROR; return code from pthread_create() is %d\n", rc);
5248       exit(-1);
5249     }
5250   }
5251   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
5252     rc = pthread_join(threads6[th], &status);
5253     if (rc) {
5254       printf("ERROR; return code from pthread_join() is %d\n", rc);
5255       exit(-1);
5256     }
5257   }
5258   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
5259     pthread_attr_destroy( &(attrs6[th]) );
5260   }
5261   (*p_minerr0) = myargstruct_array6[0].minerr0;
5262   (*p_maxerr0) = myargstruct_array6[0].maxerr0;
5263   (*p_minerr1) = myargstruct_array6[0].minerr1;
5264   (*p_maxerr1) = myargstruct_array6[0].maxerr1;
5265
5266   for(th=1;th<NUM_THREADS_USED_FOR_WORK;th++) {
5267     (*p_minerr0) =
mindouble((*p_minerr0),myargstruct_array6[th].minerr0);
5268     (*p_maxerr0) =
maxdouble((*p_maxerr0),myargstruct_array6[th].maxerr0);
5269     (*p_minerr1) =
mindouble((*p_minerr1),myargstruct_array6[th].minerr1);
5270     (*p_maxerr1) =
maxdouble((*p_maxerr1),myargstruct_array6[th].maxerr1);
5271   }
5272 }
5273
5274 void compute_minerr_maxerr(int n_examples_in_file,double*
p_minerr,double* p_maxerr) {
5275   int i;
5276   (*p_minerr) = manyruns[0].error;
5277   (*p_maxerr) = manyruns[0].error;
5278   for (i=1;i<n_examples_in_file;i++) {
5279     if (manyruns[i].error<(*p_minerr)) { (*p_minerr) =
manyruns[i].error; }
5280     if (manyruns[i].error>(*p_maxerr)) { (*p_maxerr) =
manyruns[i].error; }
5281   }
```

```
5282 }
5283
5284 int j_order_array[inputsizeinnumberofbits];
5285
5286 void shuffle_array_with_j_order() {
5287   int temp; int j1; int j2;
5288   for (temp=0;temp<inputsizeinnumberofbits*100;temp++) {
5289     j1 = random()%inputsizeinnumberofbits;
5290     j2 = random()%(inputsizeinnumberofbits-1);
5291     if (j1<=j2) { j2 = j2 + 1; }
5292     swapint(&(j_order_array[j1]),&(j_order_array[j2]));
5293   }
5294 }
5295
5296 void generate_random_array_with_j_order() {
5297   int temp;
5298   for (temp=0;temp<inputsizeinnumberofbits;temp++) {
5299     j_order_array[temp] = temp;
5300   }
5301   shuffle_array_with_j_order();
5302 }
5303
5304 int find_index_of_value_in_j_order_array(int sought_value) {
5305   int temp;
5306   for (temp=0;temp<inputsizeinnumberofbits;temp++) {
5307     if (j_order_array[temp]==sought_value) {
5308       return temp;
5309     }
5310   }
5311   printf("Error in find_index_of_value_in_j_order_array\n");
5312   exit(-1);
5313 }
5314
5315 void initialize_w0_and_b56(int n_examples_in_file) {
5316   int i; int j;
5317   p_b56_value[0] = (min_observed_time+max_observed_time)/2.0;
5318   for (j=0;j<inputsizeinnumberofbits;j++) {
5319     p_w0_value[j] = 0.0;
5320   }
5321   for (i=0;i<n_examples_in_file;i++) {
5322     manyruns[i].prediction = p_b56_value[0];
5323     manyruns[i].error = manyruns[i].prediction - manyruns[i].t;
5324   }
5325 }
5326
5327 struct myargstruct7 {
5328   int j;
5329   double delta;
5330   int loindex;
5331   int hiindex;
5332 };
5333 pthread_t threads7[NUM_THREADS_USED_FOR_WORK];
5334 pthread_attr_t attrs7[NUM_THREADS_USED_FOR_WORK];
5335 struct myargstruct7 myargstruct_array7[NUM_THREADS_USED_FOR_WORK];
5336
5337 void set_indices_in_myargstruct_array7(int th,int n_examples_in_file,int
j,double delta) {
```

```
5338   int objects_per_worker;
5339   myargstruct_array7[th].j     = j;
5340   myargstruct_array7[th].delta = delta;
5341   if (n_examples_in_file%NUM_THREADS_USED_FOR_WORK==0) {
5342     objects_per_worker = n_examples_in_file/NUM_THREADS_USED_FOR_WORK;
5343   } else {
5344     objects_per_worker = n_examples_in_file/NUM_THREADS_USED_FOR_WORK +
1;
5345   }
5346   myargstruct_array7[th].loindex = th    * objects_per_worker;
5347   myargstruct_array7[th].hiindex = (th+1)* objects_per_worker - 1;
5348   if (myargstruct_array7[th].hiindex>n_examples_in_file-1) {
5349     myargstruct_array7[th].hiindex = n_examples_in_file-1;
5350   }
5351 }
5352
5353 __attribute__((optimize("-O3"))) void* worker7(void *p) {
5354   int i;
5355   for (i=((struct myargstruct7*) p)->loindex;i<=((struct myargstruct7*)
p)->hiindex;i++) {
5356     if
(isbitonein_bitposition_in_integerarray(manyruns[i].inputtoprogram,((struct
myargstruct7*) p)->j)) {
5357       manyruns[i].prediction = manyruns[i].prediction - ((struct
myargstruct7*) p)->delta;
5358     } else {
5359       manyruns[i].prediction = manyruns[i].prediction + ((struct
myargstruct7*) p)->delta;
5360     }
5361     manyruns[i].error = manyruns[i].prediction - manyruns[i].t;
5362   }
5363   pthread_exit(NULL);
5364 }
5365
5366 void update_prediction_and_error_as_part_of_adjust_w0(int
n_examples_in_file,int j,double delta) {
5367   int rc; int th; void* status;
5368   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
5369     pthread_attr_init(&(attrs7[th]));
5370     pthread_attr_setdetachstate(&(attrs7[th]),PTHREAD_CREATE_JOINABLE);
5371     set_indices_in_myargstruct_array7(th,n_examples_in_file,j,delta);
5372     rc = pthread_create(&(threads7[th]),&(attrs7[th]),worker7,(void*)
(&(myargstruct_array7[th])));
5373     if (rc){
5374       printf("ERROR; return code from pthread_create() is %d\n", rc);
5375       exit(-1);
5376     }
5377   }
5378   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
5379     rc = pthread_join(threads7[th], &status);
5380     if (rc) {
5381       printf("ERROR; return code from pthread_join() is %d\n", rc);
5382       exit(-1);
5383     }
5384   }
5385   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
5386     pthread_attr_destroy( &(attrs7[th]) );
```

```
5387   }
5388 }
5389
5390 double minerro0_array[inputsizeinnumberofbits];
5391 double maxerro0_array[inputsizeinnumberofbits];
5392 double minerro1_array[inputsizeinnumberofbits];
5393 double maxerro1_array[inputsizeinnumberofbits];
5394
5395 void store_values(int j,double minerr0,double maxerr0,double
minerr1,double maxerr1) {
5396   minerro0_array[j] = minerr0;
5397   maxerro0_array[j] = maxerr0;
5398   minerro1_array[j] = minerr1;
5399   maxerro1_array[j] = maxerr1;
5400 }
5401
5402 void write_store_values_to_file_auxiliary_function(char* fn_prefix,int
temp,char* name,double* p_double_array) {
5403   char fn[200]; FILE* f; int j;
5404   sprintf(fn,"%s_store_values_%s_array_%d",fn_prefix,name,temp);
5405   f = fopen(fn, "w");
5406   if (f==NULL) { printf("In
write_store_values_to_file_auxiliary_function. Opening file. failed\n");
exit(-1); }
5407   for (j=0;j<inputsizeinnumberofbits;j++) {
5408     fprintf(f,"%18.15lf\n",p_double_array[j]);
5409   }
5410   fclose( f);
5411 }
5412
5413 void write_store_values_to_file(char* fn_prefix,int temp) {
5414
write_store_values_to_file_auxiliary_function(fn_prefix,temp,"minerro0",miner
ro0_array);
5415
write_store_values_to_file_auxiliary_function(fn_prefix,temp,"maxerro0",maxer
ro0_array);
5416
write_store_values_to_file_auxiliary_function(fn_prefix,temp,"minerro1",miner
ro1_array);
5417
write_store_values_to_file_auxiliary_function(fn_prefix,temp,"maxerro1",maxer
ro1_array);
5418 }
5419
5420 void setup_j_order_array_in_careful_order() {
5421   int counter; int temp_bit; int temp_integer; int j;
5422   counter = 0;
5423   for (temp_bit=0;temp_bit<32;temp_bit++) {
5424     for (temp_integer=0;temp_integer<512;temp_integer++) {
5425       j = (31-temp_bit) + temp_integer       *32; j_order_array[counter]
= j; counter = counter + 1;
5426       j = (31-temp_bit) + (1023-temp_integer)*32; j_order_array[counter]
= j; counter = counter + 1;
5427     }
5428   }
5429 }
```

```
5430
5431 void adjust_w0(int n_examples_in_file) {
5432   int temp; int i; int j; double minerr0; double maxerr0; double
minerr1; double maxerr1; double miderr0; double miderr1; double delta;
5433   setup_j_order_array_in_careful_order();
5434   for (temp=0;temp<inputsizeinnumberofbits;temp++) {
5435     j = j_order_array[temp];
5436
compute_minerr_maxerr_for_bit_for_0_and_1(n_examples_in_file,j,&minerr0,&maxe
rr0,&minerr1,&maxerr1);
5437     store_values(j,minerr0,maxerr0,minerr1,maxerr1);
5438     miderr0 = (minerr0 + maxerr0)/2.0;
5439     miderr1 = (minerr1 + maxerr1)/2.0;
5440     delta = ((miderr1 - miderr0)/2.0)*ADJUST_W0_LEARNINGRATE; // this
means that for examples with bit j=1, we should decrease prediction by delta;
for examples with bit j=0, increase by delta
5441     p_w0_value[j]  = p_w0_value[j]  - 2.0*delta;
5442     p_b56_value[0] = p_b56_value[0] +     delta;
5443
update_prediction_and_error_as_part_of_adjust_w0(n_examples_in_file,j,delta);
5444   }
5445 }
5446
5447
5448 void adjust_b56(int n_examples_in_file) {
5449   int i; double minerr; double maxerr; double delta;
5450   compute_minerr_maxerr(n_examples_in_file,&minerr,&maxerr);
5451   delta = (minerr + maxerr)/2.0;
5452   p_b56_value[0] = p_b56_value[0] - delta;
5453   for (i=0;i<n_examples_in_file;i++) {
5454     manyruns[i].prediction = manyruns[i].prediction - delta;
5455     manyruns[i].error = manyruns[i].prediction - manyruns[i].t;
5456   }
5457 }
5458
5459 double getmaxabserror(int n_examples_in_file) {
5460   int i; double maxabserror;
5461   maxabserror = fabs(manyruns[0].error);
5462   for (i=1;i<n_examples_in_file;i++) {
5463     maxabserror = maxdouble(maxabserror,fabs(manyruns[i].error));
5464   }
5465   return maxabserror;
5466 }
5467
5468 void print_statistics_when_learning(int n_examples_in_file) {
5469   int index;
5470   printf("HERE  3. max abs error = %18.15lf\n",
getmaxabserror(n_examples_in_file) );
5471   printf("HERE  4. b_b56_value[0] = %18.15lf\n", p_b56_value[0] );
5472   printf("HERE  5. p_w0_value[32767-63] = %18.15lf\n", p_w0_value[32767-
63] );
5473   printf("HERE  6. p_w0_value[32767-32] = %18.15lf\n", p_w0_value[32767-
32] );
5474   printf("HERE  7. p_w0_value[32767-31] = %18.15lf\n", p_w0_value[32767-
31] );
5475   printf("HERE  8. p_w0_value[32767]    = %18.15lf\n", p_w0_value[32767]
);
```

```
5476   index = find_index_of_value_in_j_order_array(32767-63); printf("HERE
9. j_order_array[%d] = %d\n", index, j_order_array[index] );
5477   index = find_index_of_value_in_j_order_array(32767-32); printf("HERE
10. j_order_array[%d] = %d\n", index, j_order_array[index] );
5478   index = find_index_of_value_in_j_order_array(32767-31); printf("HERE
11. j_order_array[%d] = %d\n", index, j_order_array[index] );
5479   index = find_index_of_value_in_j_order_array(32767);    printf("HERE
12. j_order_array[%d] = %d\n", index, j_order_array[index] );
5480 }
5481
5482 void
learn_w0_b56_through_sequential_bit_by_bit_learning_minimize_max_abs_error(in
t n_examples_in_file,char* fn_prefix) {
5483   int temp;
5484   generate_random_array_with_j_order();
5485   initialize_w0_and_b56(n_examples_in_file);
5486   printf("HERE  1. max abs error = %18.15lf\n",
getmaxabserror(n_examples_in_file) );
5487   for (temp=0;temp<ADJUST_W0_B56_NEPOCHS;temp++) {
5488     printf("HERE  2. temp = %d\n", temp );
5489     shuffle_array_with_j_order();
5490     adjust_w0(n_examples_in_file);
5491     adjust_b56(n_examples_in_file);
5492     print_statistics_when_learning(n_examples_in_file);
5493     write_store_values_to_file(fn_prefix,temp);
5494   }
5495 }
5496
5497 void write_w0_to_textfile(char* fn_prefix) {
5498   char fn[200]; FILE* f; int j;
5499   sprintf(fn,"%s_w0.txt",fn_prefix);
5500   f = fopen(fn, "w" ); if (f==NULL) { printf("Error in
write_w0_to_textfile\n"); fflush(stdout); exit(-1); }
5501   for (j=0;j<inputsizeinnumberofbits;j++) {
5502     fprintf(f,"%18.15lf\n", p_w0_value[j]);
5503   }
5504   fclose(f);
5505 }
5506 void write_b56_to_textfile(char* fn_prefix) {
5507   char fn[200]; FILE* f;
5508   sprintf(fn,"%s_b56.txt",fn_prefix);
5509   f = fopen(fn, "w" ); if (f==NULL) { printf("Error in
write_b56_to_textfile\n"); fflush(stdout); exit(-1); }
5510   fprintf(f,"%18.15lf\n", p_b56_value[0]);
5511   fclose(f);
5512 }
5513 void write_w0_b56_to_textfile(char* fn_prefix) {
5514   write_w0_to_textfile(fn_prefix);
5515   write_b56_to_textfile(fn_prefix);
5516 }
5517
5518 void read_w0_from_textfile(char* fn_prefix) {
5519   char fn[200]; FILE* f; int j;
5520   sprintf(fn,"%s_w0.txt",fn_prefix);
5521   // f = fopen(fn, "r" ); if (f==NULL) { printf("Error in
read_w0_from_textfile %s\n", fn); fflush(stdout); exit(-1); }
```

```
5522   f = fopen(fn, "r" ); if (f==NULL) { printf("Error in
read_w0_from_textfile\n"); fflush(stdout); exit(-1); }
5523   for (j=0;j<inputsizeinnumberofbits;j++) {
5524     fscanf(f,"%lf", &(p_w0_value[j]));
5525   }
5526   fclose(f);
5527 }
5528 void read_b56_from_textfile(char* fn_prefix) {
5529   char fn[200]; FILE* f;
5530   sprintf(fn,"%s_b56.txt",fn_prefix);
5531   f = fopen(fn, "r" ); if (f==NULL) { printf("Error in
read_b56_from_textfile\n"); fflush(stdout); exit(-1); }
5532   fscanf(f,"%lf", &(p_b56_value[0]));
5533   fclose(f);
5534 }
5535 void read_w0_b56_from_textfile(char* fn_prefix) {
5536   read_w0_from_textfile(fn_prefix);
5537   read_b56_from_textfile(fn_prefix);
5538 }
5539
5540 struct myargstruct8 {
5541   struct basisfunction a_basisfunction;
5542   int loindex;
5543   int hiindex;
5544   double minerr0; double maxerr0; double minerr1; double maxerr1;
5545 };
5546 pthread_t threads8[NUM_THREADS_USED_FOR_WORK];
5547 pthread_attr_t attrs8[NUM_THREADS_USED_FOR_WORK];
5548 struct myargstruct8 myargstruct_array8[NUM_THREADS_USED_FOR_WORK];
5549
5550 void set_indices_in_myargstruct_array8(int th,int
n_examples_in_file,struct basisfunction a_basisfunction) {
5551   int objects_per_worker;
5552   myargstruct_array8[th].a_basisfunction = a_basisfunction;
5553   if (n_examples_in_file%NUM_THREADS_USED_FOR_WORK==0) {
5554     objects_per_worker = n_examples_in_file/NUM_THREADS_USED_FOR_WORK;
5555   } else {
5556     objects_per_worker = n_examples_in_file/NUM_THREADS_USED_FOR_WORK +
1;
5557   }
5558   myargstruct_array8[th].loindex = th    * objects_per_worker;
5559   myargstruct_array8[th].hiindex = (th+1)* objects_per_worker - 1;
5560   if (myargstruct_array8[th].hiindex>n_examples_in_file-1) {
5561     myargstruct_array8[th].hiindex = n_examples_in_file-1;
5562   }
5563 }
5564
5565 __attribute__((optimize("-O3"))) void* worker8(void *p) {
5566   int i; int bit_equals_0_result_available; int
bit_equals_1_result_available;
5567   bit_equals_0_result_available = 0;
5568   bit_equals_1_result_available = 0;
5569   for (i=((struct myargstruct8*) p)->loindex;i<=((struct myargstruct8*)
p)->hiindex;i++) {
5570     if (evaluate_basis_on_example_v2_seq(&(((struct myargstruct8*) p)-
>a_basisfunction),i)) {
5571       if (bit_equals_1_result_available) {
```

```
5572          if (manyruns[i].error<((struct myargstruct8*) p)->minerr1) {
((struct myargstruct8*) p)->minerr1 = manyruns[i].error; }
5573          if (manyruns[i].error>((struct myargstruct8*) p)->maxerr1) {
((struct myargstruct8*) p)->maxerr1 = manyruns[i].error; }
5574        } else {
5575          ((struct myargstruct8*) p)->minerr1 = manyruns[i].error;
5576          ((struct myargstruct8*) p)->maxerr1 = manyruns[i].error;
5577          bit_equals_1_result_available = 1;
5578        }
5579      } else {
5580        if (bit_equals_0_result_available) {
5581          if (manyruns[i].error<((struct myargstruct8*) p)->minerr0) {
((struct myargstruct8*) p)->minerr0 = manyruns[i].error; }
5582          if (manyruns[i].error>((struct myargstruct8*) p)->maxerr0) {
((struct myargstruct8*) p)->maxerr0 = manyruns[i].error; }
5583        } else {
5584          ((struct myargstruct8*) p)->minerr0 = manyruns[i].error;
5585          ((struct myargstruct8*) p)->maxerr0 = manyruns[i].error;
5586          bit_equals_0_result_available = 1;
5587        }
5588      }
5589    }
5590    if (!bit_equals_0_result_available) {
5591      ((struct myargstruct8*) p)->minerr0 = 0.0;
5592      ((struct myargstruct8*) p)->maxerr0 = 0.0;
5593    }
5594    if (!bit_equals_1_result_available) {
5595      ((struct myargstruct8*) p)->minerr1 = 0.0;
5596      ((struct myargstruct8*) p)->maxerr1 = 0.0;
5597    }
5598    pthread_exit(NULL);
5599  }
5600
5601  void compute_minerr_maxerr_for_bit_for_0_and_1_for_basis_function(int
n_examples_in_file,struct basisfunction a_basisfunction,double*
p_minerr0,double* p_maxerr0,double* p_minerr1,double* p_maxerr1) {
5602    int rc; int th; void* status;
5603    for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
5604      pthread_attr_init(&(attrs8[th]));
5605      pthread_attr_setdetachstate(&(attrs8[th]),PTHREAD_CREATE_JOINABLE);
5606
set_indices_in_myargstruct_array8(th,n_examples_in_file,a_basisfunction);
5607      rc = pthread_create(&(threads8[th]),&(attrs8[th]),worker8,(void*)
(&(myargstruct_array8[th])));
5608      if (rc){
5609        printf("ERROR; return code from pthread_create() is %d\n", rc);
5610        exit(-1);
5611      }
5612    }
5613    for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
5614      rc = pthread_join(threads8[th], &status);
5615      if (rc) {
5616        printf("ERROR; return code from pthread_join() is %d\n", rc);
5617        exit(-1);
5618      }
5619    }
5620    for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
```

```
5621      pthread_attr_destroy( &(attrs8[th]) );
5622    }
5623    (*p_minerr0) = myargstruct_array8[0].minerr0;
5624    (*p_maxerr0) = myargstruct_array8[0].maxerr0;
5625    (*p_minerr1) = myargstruct_array8[0].minerr1;
5626    (*p_maxerr1) = myargstruct_array8[0].maxerr1;
5627    for(th=1;th<NUM_THREADS_USED_FOR_WORK;th++) {
5628      (*p_minerr0) =
mindouble((*p_minerr0),myargstruct_array8[th].minerr0);
5629      (*p_maxerr0) =
maxdouble((*p_maxerr0),myargstruct_array8[th].maxerr0);
5630      (*p_minerr1) =
mindouble((*p_minerr1),myargstruct_array8[th].minerr1);
5631      (*p_maxerr1) =
maxdouble((*p_maxerr1),myargstruct_array8[th].maxerr1);
5632    }
5633  }
5634
5635  double compute_delta_minimize_max_abserror(int n_examples_in_file,struct
basisfunction a_basisfunction) {
5636    double minerr0; double maxerr0; double minerr1; double maxerr1; double
miderr0; double miderr1; double delta;
5637
compute_minerr_maxerr_for_bit_for_0_and_1_for_basis_function(n_examples_in_fi
le,a_basisfunction,&minerr0,&maxerr0,&minerr1,&maxerr1);
5638    miderr0 = (minerr0 + maxerr0)/2.0;
5639    miderr1 = (minerr1 + maxerr1)/2.0;
5640    delta = (miderr1 - miderr0)/2.0; // this means that for examples with
basisfunction=1, we should decrease prediction by delta; for examples with
basisfunction=0, increase by delta
5641    return delta;
5642  }
5643
5644  struct myargstruct9 {
5645    struct basisfunction a_basisfunction;
5646    double delta;
5647    int loindex;
5648    int hiindex;
5649  };
5650  pthread_t threads9[NUM_THREADS_USED_FOR_WORK];
5651  pthread_attr_t attrs9[NUM_THREADS_USED_FOR_WORK];
5652  struct myargstruct9 myargstruct_array9[NUM_THREADS_USED_FOR_WORK];
5653
5654  void set_indices_in_myargstruct_array9(int th,int
n_examples_in_file,struct basisfunction a_basisfunction,double delta) {
5655    int objects_per_worker;
5656    myargstruct_array9[th].a_basisfunction     = a_basisfunction;
5657    myargstruct_array9[th].delta               = delta;
5658    if (n_examples_in_file%NUM_THREADS_USED_FOR_WORK==0) {
5659      objects_per_worker = n_examples_in_file/NUM_THREADS_USED_FOR_WORK;
5660    } else {
5661      objects_per_worker = n_examples_in_file/NUM_THREADS_USED_FOR_WORK +
1;
5662    }
5663    myargstruct_array9[th].loindex = th    * objects_per_worker;
5664    myargstruct_array9[th].hiindex = (th+1)* objects_per_worker - 1;
5665    if (myargstruct_array9[th].hiindex>n_examples_in_file-1) {
```

```
5666      myargstruct_array9[th].hiindex = n_examples_in_file-1;
5667    }
5668  }
5669
5670  __attribute__((optimize("-O3"))) void* worker9(void *p) {
5671    int i;
5672    for (i=((struct myargstruct9*) p)->loindex;i<=((struct myargstruct9*)
p)->hiindex;i++) {
5673      if (evaluate_basis_on_example_v2_seq(&(((struct myargstruct9*) p)-
>a_basisfunction),i)) {
5674        manyruns[i].prediction = manyruns[i].prediction - ((struct
myargstruct9*) p)->delta;
5675      } else {
5676        manyruns[i].prediction = manyruns[i].prediction + ((struct
myargstruct9*) p)->delta;
5677      }
5678      manyruns[i].error = manyruns[i].prediction - manyruns[i].t;
5679    }
5680    pthread_exit(NULL);
5681  }
5682
5683  void update_prediction_and_error_as_part_of_adding_basis_functions(int
n_examples_in_file,struct basisfunction a_basisfunction,double delta) {
5684    int rc; int th; void* status;
5685    for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
5686      pthread_attr_init(&(attrs9[th]));
5687      pthread_attr_setdetachstate(&(attrs9[th]),PTHREAD_CREATE_JOINABLE);
5688
set_indices_in_myargstruct_array9(th,n_examples_in_file,a_basisfunction,delta
);
5689      rc = pthread_create(&(threads9[th]),&(attrs9[th]),worker9,(void*)
(&(myargstruct_array9[th])));
5690      if (rc){
5691        printf("ERROR; return code from pthread_create() is %d\n", rc);
5692        exit(-1);
5693      }
5694    }
5695    for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
5696      rc = pthread_join(threads9[th], &status);
5697      if (rc) {
5698        printf("ERROR; return code from pthread_join() is %d\n", rc);
5699        exit(-1);
5700      }
5701    }
5702    for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
5703      pthread_attr_destroy( &(attrs9[th]) );
5704    }
5705  }
5706
5707  void find_suitable_basis_functions_minimize_max_abserror(char*
fn_prefix,int n_examples_in_file) {
5708    int basisfunctionindex; struct basisfunction best_basisfunction;
struct basisfunction new_basisfunction; int iterator; double max_abs_delta;
int max_abs_delta_has_been_assigned; double delta;
5709    for
(basisfunctionindex=0;basisfunctionindex<NBASISFUNCTIONS;basisfunctionindex++
) {
```

```
5710      max_abs_delta = -1.0;
5711      max_abs_delta_has_been_assigned = 0;
5712      for (iterator=0;iterator<SELECT_BESTBASISFUNCTION_AMONG;iterator++)
{
5713        new_basisfunction = generate_new_basisfunction();
5714        while (is_basisfunction_in(basisfunctionindex,new_basisfunction))
{
5715          new_basisfunction = generate_new_basisfunction();
5716        }
5717        delta =
compute_delta_minimize_max_abserror(n_examples_in_file,new_basisfunction); //
this means that for examples with basisfunction=1, we should decrease
prediction by delta; for examples with basisfunction=0, increase by delta
5718
5719        if (max_abs_delta_has_been_assigned==0) {
5720          max_abs_delta = fabs(delta);
5721          best_basisfunction = new_basisfunction;
5722          max_abs_delta_has_been_assigned = 1;
5723        } else {
5724          if (max_abs_delta<fabs(delta)) {
5725            max_abs_delta = fabs(delta);
5726            best_basisfunction = new_basisfunction;
5727          }
5728        }
5729      }
5730      basisfunctions[basisfunctionindex] = best_basisfunction;
5731      basisfunctions[basisfunctionindex].coefficient = (-2.0)*delta;
5732      p_b56_value[0]                      = p_b56_value[0] + delta;
5733
update_prediction_and_error_as_part_of_adding_basis_functions(n_examples_in_f
ile,best_basisfunction,delta);
5734    }
5735  }
5736
5737  struct myargstruct10 {
5738    int loindex;
5739    int hiindex;
5740  };
5741  pthread_t threads10[NUM_THREADS_USED_FOR_WORK];
5742  pthread_attr_t attrs10[NUM_THREADS_USED_FOR_WORK];
5743  struct myargstruct10 myargstruct_array10[NUM_THREADS_USED_FOR_WORK];
5744
5745  void set_indices_in_myargstruct_array10(int th,int n_examples_in_file) {
5746    int objects_per_worker;
5747    if (n_examples_in_file%NUM_THREADS_USED_FOR_WORK==0) {
5748      objects_per_worker = n_examples_in_file/NUM_THREADS_USED_FOR_WORK;
5749    } else {
5750      objects_per_worker = n_examples_in_file/NUM_THREADS_USED_FOR_WORK +
1;
5751    }
5752    myargstruct_array10[th].loindex = th    * objects_per_worker;
5753    myargstruct_array10[th].hiindex = (th+1)* objects_per_worker - 1;
5754    if (myargstruct_array10[th].hiindex>n_examples_in_file-1) {
5755      myargstruct_array10[th].hiindex = n_examples_in_file-1;
5756    }
5757  }
5758
```

```
5759 __attribute__((optimize("-O3"))) void* worker10(void *p) {
5760   int i;
5761   for (i=((struct myargstruct10*) p)->loindex;i<=((struct
myargstruct10*) p)->hiindex;i++) {
5762     // manyruns[i].prediction =
compute_prediction_on_example_w0_b56_basis_functions( i);
5763     manyruns[i].prediction =
compute_prediction_on_example_w0_b56_basis_functions_seq( i);
5764     manyruns[i].error = manyruns[i].prediction - manyruns[i].t;
5765   }
5766   pthread_exit(NULL);
5767 }
5768
5769 void
compute_error_and_prediction_based_on_w0_b56_and_basis_functions(int
n_examples_in_file) { // note that that caller may pass n_selected_examples
as parameter
5770   int rc; int th; void* status;
5771   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
5772     pthread_attr_init(&(attrs10[th]));
5773     pthread_attr_setdetachstate(&(attrs10[th]),PTHREAD_CREATE_JOINABLE);
5774     set_indices_in_myargstruct_array10(th,n_examples_in_file);
5775     rc = pthread_create(&(threads10[th]),&(attrs10[th]),worker10,(void*)
(&(myargstruct_array10[th])));
5776     if (rc){
5777       printf("ERROR; return code from pthread_create() is %d\n", rc);
5778       exit(-1);
5779     }
5780   }
5781   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
5782     rc = pthread_join(threads10[th], &status);
5783     if (rc) {
5784       printf("ERROR; return code from pthread_join() is %d\n", rc);
5785       exit(-1);
5786     }
5787   }
5788   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
5789     pthread_attr_destroy( &(attrs10[th]) );
5790   }
5791 }
5792
5793 int find_WCET_v2(char* fn_with_executiontimemeasurements_inputs,char*
fn_with_executiontimemeasurements_times,char* fn_prefix,int programid,
5794   int n_examples_in_file) {
5795   char fn[2000]; char fn2[2000]; double t0; double t1; FILE* f;
5796   char temp_fn_prefix[200];
5797
allocate_memory_for_predictedworstcaseinput_phase0_and_predictedworstcaseinpu
t_phase1();
5798   allocate_memory_for_learnable_weights_v2(); // this allocates memory
for w0 and b56 which we need; we don't need the others that are allocared by
this func
5799   allocate_memory_for_manyruns_v2(n_examples_in_file);
5800
read_captured_data_from_disk_two_explicit_fn(fn_with_executiontimemeasurement
s_inputs,fn_with_executiontimemeasurements_times,n_examples_in_file);
```

```
5801    printf("n_examples_in_file = %d\n",n_examples_in_file);
fflush(stdout);
5802
calculate_min_max_average_executiontime_from_manyruns(n_examples_in_file);
5803
5804
learn_w0_b56_through_sequential_bit_by_bit_learning_minimize_max_abs_error(n_
examples_in_file,fn_prefix);
5805    write_w0_b56_to_textfile(fn_prefix);
5806    read_w0_b56_from_textfile(fn_prefix);
5807    compute_error_and_prediction_based_on_w0_b56(n_examples_in_file);
5808    sprintf(temp_fn_prefix,"%s_for_affine_model",fn_prefix);
5809    write_all_predictions_to_file(temp_fn_prefix,n_examples_in_file);
5810    write_all_errors_to_file(temp_fn_prefix,n_examples_in_file);
5811
write_summary_of_predictions_and_errors_to_files(temp_fn_prefix,n_examples_in
_file);
5812
5813    exit(-1);
5814
5815
find_WCET_input_from_prediction_model_w0_b56(fn_prefix,predictedworstcaseinpu
t,&predicted_time_predictedworstcaseinput);
5816
copy_inputtoprogram(predictedworstcaseinput_phase0,predictedworstcaseinput);
5817    predicted_time_predictedworstcaseinput_phase0 =
predicted_time_predictedworstcaseinput;
5818
5819
find_suitable_basis_functions_minimize_max_abserror(fn_prefix,n_examples_in_f
ile);
5820    print_basis_functions_to_file(fn_prefix);
5821    read_basis_functions_from_file(fn_prefix);
5822
compute_error_and_prediction_based_on_w0_b56_and_basis_functions(n_examples_i
n_file);
5823
sprintf(temp_fn_prefix,"%s_for_affine_model_and_basis_functions",fn_prefix);
5824    write_all_predictions_to_file(temp_fn_prefix,n_examples_in_file);
5825    write_all_errors_to_file(temp_fn_prefix,n_examples_in_file);
5826
write_summary_of_predictions_and_errors_to_files(temp_fn_prefix,n_examples_in
_file);
5827
5828
find_WCET_input_from_prediction_model_w0_b56_w1_b1_w5(fn_prefix,predictedwors
tcaseinput,&predicted_time_predictedworstcaseinput); // THIS NEEDS TO BE
REWRITTEN TO TAKE INTO ACOUNT BASIS FUNCTIONS
5829
5830
copy_inputtoprogram(predictedworstcaseinput_phase1,predictedworstcaseinput);
5831    predicted_time_predictedworstcaseinput_phase1 =
predicted_time_predictedworstcaseinput;
5832
5833    printf("Running with obtained worst-case input\n");
5834    sleep(10);
5835    setprocessoraffinity_to_allow_just_a_single_processor_proc0();
```

```
5836   sleep(60);
5837
run_program_with_two_different_inputs_dont_use_filesystem_run_X_times(predict
edworstcaseinput_phase0,predictedworstcaseinput_phase1,programid,100,&t0,&t1)
;
5838
5839   sprintf(fn, "%s_0_predictedworstcaseinput.dat",        fn_prefix);
5840   sprintf(fn2,"%s_0_predictedworstcaseexecutiontime.dat",fn_prefix);
5841
write_input_and_execution_time_to_files(fn,fn2,predictedworstcaseinput_phase0
,t0);
5842
sprintf(fn,"%s_0_predictedworstcaseexecutiontime_notrun.dat",fn_prefix); f =
fopen(fn, "w+"); fprintf( f, "%18.12lf\n",
predicted_time_predictedworstcaseinput_phase0); fclose( f);
5843
5844   sprintf(fn, "%s_1_predictedworstcaseinput.dat",        fn_prefix);
5845   sprintf(fn2,"%s_1_predictedworstcaseexecutiontime.dat",fn_prefix);
5846
write_input_and_execution_time_to_files(fn,fn2,predictedworstcaseinput_phase1
,t1);
5847
sprintf(fn,"%s_1_predictedworstcaseexecutiontime_notrun.dat",fn_prefix); f =
fopen(fn, "w+"); fprintf( f, "%18.12lf\n",
predicted_time_predictedworstcaseinput_phase1); fclose( f);
5848
5849   setprocessoraffinity_to_allow_all_processors(); // we don't need to
restore this but it does not hurt
5850
5851   free_memory_for_learnable_weights_v2(); // we only need w0 and b56
5852   free_memory_for_manyruns_v2(n_examples_in_file);
5853
free_memory_for_predictedworstcaseinput_phase0_and_predictedworstcaseinput_ph
ase1();
5854 }
5855
5856 double* t_values_used_for_cmpfunc_sort_examples_in_ascending_order_of_t;
5857 int cmpfunc_sort_examples_in_ascending_order_of_t(const void * a, const
void * b) {
5858   double t_a; double t_b;
5859   t_a =
t_values_used_for_cmpfunc_sort_examples_in_ascending_order_of_t[*((int*) a)];
5860   t_b =
t_values_used_for_cmpfunc_sort_examples_in_ascending_order_of_t[*((int*) b)];
5861   if (t_a<t_b) {
5862     return -1;
5863   } else {
5864     if (t_a==t_b) {
5865       return 0;
5866     } else {
5867       return 1;
5868     }
5869   }
5870 }
5871 void sort_all_indices_of_examples_in_ascending_order_of_t(int
n_examples_in_file,int* all_indices_of_examples,double* t_values) {
```

```
5872   t_values_used_for_cmpfunc_sort_examples_in_ascending_order_of_t =
t_values;
5873   qsort(all_indices_of_examples, n_examples_in_file, sizeof(int),
cmpfunc_sort_examples_in_ascending_order_of_t);
5874 }
5875
5876 void read_all_t_values_from_file(char* fn,int n_examples_in_file,double*
t_values) {
5877   FILE* f; int i;
5878   f = fopen(fn, "rb"); if (f==NULL) { printf("Error in
read_all_t_values_from_file. Opening file. failed\n"); exit(-1); }
5879   for (i=0;i<n_examples_in_file;i++) {
5880     fread(&(t_values[i]),sizeof(double),1,f);
5881   }
5882   fclose( f);
5883 }
5884
5885 int find_example_closest_to(double target_t,int n_examples_in_file,int*
all_indices_of_examples,double* t_values) {
5886   int lb_index; int ub_index;
5887   double t_lb_index; double t_ub_index;
5888   int index_middle;
5889   double t_middle;
5890   lb_index = 0;
5891   ub_index = n_examples_in_file-1;
5892   t_lb_index = t_values[all_indices_of_examples[lb_index]];
5893   t_ub_index = t_values[all_indices_of_examples[ub_index]];
5894   while (lb_index<ub_index) {
5895     if (lb_index+1==ub_index) {
5896       t_middle = (t_lb_index+t_ub_index)/2;
5897       if (target_t<t_middle) {
5898         ub_index = ub_index - 1;
5899       } else {
5900         lb_index = lb_index + 1;
5901       }
5902     } else {
5903       index_middle = (lb_index+ub_index)/2;
5904       t_middle = t_values[all_indices_of_examples[index_middle]];
5905       if (target_t<t_middle) {
5906         ub_index = index_middle;
5907         t_ub_index = t_middle;
5908       } else {
5909         lb_index = index_middle;
5910         t_lb_index = t_middle;
5911       }
5912     }
5913   }
5914   return all_indices_of_examples[lb_index];
5915 }
5916
5917 int cmpfunc_sort_selected_indices_of_examples_in_ascending_order(const
void * a, const void * b) {
5918   int* p_a;
5919   int* p_b;
5920   p_a = (int*) a;
5921   p_b = (int*) b;
5922   if ((*p_a) < (*p_b)) {
```

```
5923      return -1;
5924    } else {
5925      if ((*p_a) == (*p_b)) {
5926        return 0;
5927      } else {
5928        return 1;
5929      }
5930    }
5931  }
5932
5933  void sort_selected_indices_of_examples_in_ascending_order(int
n_examples_selected,int* selected_indices_of_examples) {
5934    qsort(selected_indices_of_examples, n_examples_selected, sizeof(int),
cmpfunc_sort_selected_indices_of_examples_in_ascending_order);
5935  }
5936
5937  #define SELECT_LOW_RATIO    0.45
5938  #define SELECT_MIDDLE_RATIO 0.10
5939  #define SELECT_HIGH_RATIO   0.45
5940
5941  void fill_many_examples_from_files(char*
fn_with_executiontimemeasurements_inputs,char*
fn_with_executiontimemeasurements_times,int n_examples_in_file,int
n_examples_selected) {
5942    int*    all_indices_of_examples; // this will be sorted wrt to
t_values;
5943    double* t_values;
5944    int*    selected_indices_of_examples;
5945    int i; double slope; FILE* f_inputs; FILE* f_times;
5946    all_indices_of_examples      = malloc(n_examples_in_file*sizeof(int));
if (all_indices_of_examples==NULL)      { printf("malloc failure
fill_many_examples_from_files. all_indices_of_examples\n"); exit(-1); }
5947    t_values                     =
malloc(n_examples_in_file*sizeof(double)); if (t_values==NULL)
{ printf("malloc failure fill_many_examples_from_files. t_values\n"); exit(-
1); }
5948    selected_indices_of_examples =
malloc(n_examples_selected*sizeof(int));   if
(selected_indices_of_examples==NULL) { printf("malloc failure
fill_many_examples_from_files. selected_indices_of_examples\n"); exit(-1); }
5949    for (i=0;i<n_examples_in_file;i++) { all_indices_of_examples[i] = i; }
5950
read_all_t_values_from_file(fn_with_executiontimemeasurements_times,n_example
s_in_file,t_values);
5951
sort_all_indices_of_examples_in_ascending_order_of_t(n_examples_in_file,all_i
ndices_of_examples,t_values);
5952    for (i=0;i<=n_examples_selected-1;i++) {
5953      // if (i<=((n_examples_selected/4)-1)) {
5954      if (i<=(SELECT_LOW_RATIO*n_examples_selected)-1) {
5955        selected_indices_of_examples[i] = i;
5956      } else {
5957        // if ((3*n_examples_selected/4)<=i) {
5958        if ((SELECT_LOW_RATIO+SELECT_MIDDLE_RATIO)*n_examples_selected<=i)
{
5959          selected_indices_of_examples[i] = n_examples_in_file+i-
n_examples_selected;
```

```
5960        } else {
5961          // slope = (n_examples_in_file-(n_examples_selected/2.0)) /
(n_examples_selected/2.0);
5962          slope = (n_examples_in_file-
((SELECT_LOW_RATIO+SELECT_HIGH_RATIO)*n_examples_selected)) /
(SELECT_MIDDLE_RATIO*n_examples_selected);
5963          selected_indices_of_examples[i] =
SELECT_LOW_RATIO*n_examples_selected + (i-
SELECT_LOW_RATIO*n_examples_selected)*slope;
5964        }
5965      }
5966    }
5967    for (i=1;i<=n_examples_selected-1;i++) {
5968      if (selected_indices_of_examples[i]<=selected_indices_of_examples[i-
1]) {
5969        // printf("Error in creating selected_indices. Error\n"); exit(-
1); // based on how selected_indices are created above, this will not happen
5970        printf("Error in creating selected_indices. Error\n");
5971        printf("i = %d\n", i);
5972        printf("selected_indices_of_examples[i]   = %d\n",
selected_indices_of_examples[i]   );
5973        printf("selected_indices_of_examples[i-1] = %d\n",
selected_indices_of_examples[i-1] );
5974        exit(-1);
5975      }
5976    }
5977    for (i=0;i<=n_examples_selected-1;i++) {
5978      selected_indices_of_examples[i] =
all_indices_of_examples[selected_indices_of_examples[i]];
5979    }
5980
sort_selected_indices_of_examples_in_ascending_order(n_examples_selected,sele
cted_indices_of_examples);
5981    f_inputs = fopen(fn_with_executiontimemeasurements_inputs, "rb"); if
(f_inputs==NULL) { printf("Error opening file. i
%s\n",fn_with_executiontimemeasurements_inputs); exit(-1); }
5982    f_times  = fopen(fn_with_executiontimemeasurements_times,  "rb"); if
(f_times==NULL)  { printf("Error opening file. t
%s\n",fn_with_executiontimemeasurements_times ); exit(-1); }
5983    for (i=0;i<n_examples_selected;i++) {
5984      if (fseek(f_inputs,
((long)selected_indices_of_examples[i])*getinputtoprogram_size_in_number_of_b
ytes(),SEEK_SET)!=0) { printf("Error doing seek in file. i"); exit(-1); }
5985      if (fseek(f_times,
((long)selected_indices_of_examples[i])*sizeof(double),
SEEK_SET)!=0) { printf("Error doing seek in file. t"); exit(-1); }
5986      if
(fread(manyruns[i].inputtoprogram,getinputtoprogram_size_in_number_of_bytes()
,1,f_inputs)!=1) { printf("Error reading file. i"); exit(-1); }
5987      if (fread(&(manyruns[i].t),          sizeof(double),
1,f_times )!=1) { printf("Error reading file. t"); exit(-1); }
5988    }
5989    fclose(f_times);
5990    fclose(f_inputs);
5991    free(selected_indices_of_examples);
5992    free(t_values);
5993    free(all_indices_of_examples);
```

```
5994 }
5995
5996 double compute_avg_execution_time(int n_examples_selected) {
5997   int i; double sum;
5998   sum = manyruns[0].t;
5999   for (i=1;i<n_examples_selected;i++) {
6000     sum = sum + manyruns[i].t;
6001   }
6002   return sum/n_examples_selected;
6003 }
6004
6005 void train_affine_model_initialize_w0_and_b56(int n_examples_selected) {
6006   int i; int j;
6007   p_b56_value[0] = compute_avg_execution_time(n_examples_selected);
6008   for (j=0;j<inputsizeinnumberofbits;j++) {
6009     p_w0_value[j] = 0.0;
6010   }
6011   for (i=0;i<n_examples_selected;i++) {
6012     manyruns[i].prediction = p_b56_value[0];
6013     manyruns[i].error = manyruns[i].prediction - manyruns[i].t;
6014   }
6015 }
6016
6017 struct myargstruct11 {
6018   int j;
6019   int loindex;
6020   int hiindex;
6021   double sumerr0; int count0; double sumerr1; int count1;
6022 };
6023 pthread_t threads11[NUM_THREADS_USED_FOR_WORK];
6024 pthread_attr_t attrs11[NUM_THREADS_USED_FOR_WORK];
6025 struct myargstruct11 myargstruct_array11[NUM_THREADS_USED_FOR_WORK];
6026
6027 void set_indices_in_myargstruct_array11(int th,int
n_examples_selected,int j) {
6028   int objects_per_worker;
6029   myargstruct_array11[th].j = j;
6030   if (n_examples_selected%NUM_THREADS_USED_FOR_WORK==0) {
6031     objects_per_worker = n_examples_selected/NUM_THREADS_USED_FOR_WORK;
6032   } else {
6033     objects_per_worker = n_examples_selected/NUM_THREADS_USED_FOR_WORK +
1;
6034   }
6035   myargstruct_array11[th].loindex = th    * objects_per_worker;
6036   myargstruct_array11[th].hiindex = (th+1)* objects_per_worker - 1;
6037   if (myargstruct_array11[th].hiindex>n_examples_selected-1) {
6038     myargstruct_array11[th].hiindex = n_examples_selected-1;
6039   }
6040 }
6041
6042 __attribute__((optimize("-O3"))) void* worker11(void *p) {
6043   int i;
6044   ((struct myargstruct11*) p)->sumerr0 = 0.0;
6045   ((struct myargstruct11*) p)->count0  = 0;
6046   ((struct myargstruct11*) p)->sumerr1 = 0.0;
6047   ((struct myargstruct11*) p)->count1  = 0;
```

```
6048   for (i=((struct myargstruct11*) p)->loindex;i<=((struct
myargstruct11*) p)->hiindex;i++) {
6049     if
(isbitonein_bitposition_in_integerarray(manyruns[i].inputtoprogram,((struct
myargstruct11*) p)->j)) {
6050       ((struct myargstruct11*) p)->sumerr1 = ((struct myargstruct11*)
p)->sumerr1 + manyruns[i].error;
6051       ((struct myargstruct11*) p)->count1  = ((struct myargstruct11*)
p)->count1  + 1;
6052     } else {
6053       ((struct myargstruct11*) p)->sumerr0 = ((struct myargstruct11*)
p)->sumerr0 + manyruns[i].error;
6054       ((struct myargstruct11*) p)->count0  = ((struct myargstruct11*)
p)->count0  + 1;
6055     }
6056   }
6057   pthread_exit(NULL);
6058 }
6059
6060 void train_affine_model_compute_sumerr_count_for_bit_for_0_and_1(int
n_examples_selected,int j,double* p_sumerr0,int* p_count0,double*
p_sumerr1,int* p_count1) {
6061   int rc; int th; void* status;
6062   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
6063     pthread_attr_init(&(attrs11[th]));
6064     pthread_attr_setdetachstate(&(attrs11[th]),PTHREAD_CREATE_JOINABLE);
6065     set_indices_in_myargstruct_array11(th,n_examples_selected,j);
6066     rc = pthread_create(&(threads11[th]),&(attrs11[th]),worker11,(void*)
(&(myargstruct_array11[th])));
6067     if (rc){
6068       printf("ERROR; return code from pthread_create() is %d\n", rc);
6069       exit(-1);
6070     }
6071   }
6072   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
6073     rc = pthread_join(threads11[th], &status);
6074     if (rc) {
6075       printf("ERROR; return code from pthread_join() is %d\n", rc);
6076       exit(-1);
6077     }
6078   }
6079   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
6080     pthread_attr_destroy( &(attrs11[th]) );
6081   }
6082   (*p_sumerr0) = myargstruct_array11[0].sumerr0;
6083   (*p_count0)  = myargstruct_array11[0].count0;
6084   (*p_sumerr1) = myargstruct_array11[0].sumerr1;
6085   (*p_count1)  = myargstruct_array11[0].count1;
6086   for(th=1;th<NUM_THREADS_USED_FOR_WORK;th++) {
6087     (*p_sumerr0) = (*p_sumerr0) + myargstruct_array11[th].sumerr0;
6088     (*p_count0)  = (*p_count0)  + myargstruct_array11[th].count0;
6089     (*p_sumerr1) = (*p_sumerr1) + myargstruct_array11[th].sumerr1;
6090     (*p_count1)  = (*p_count1)  + myargstruct_array11[th].count1;
6091   }
6092 }
6093
6094 struct myargstruct12 {
```

```
6095   int j;
6096   double delta_w0_value_j;
6097   double delta_b56_value;
6098   int loindex;
6099   int hiindex;
6100 };
6101 pthread_t threads12[NUM_THREADS_USED_FOR_WORK];
6102 pthread_attr_t attrs12[NUM_THREADS_USED_FOR_WORK];
6103 struct myargstruct12 myargstruct_array12[NUM_THREADS_USED_FOR_WORK];
6104
6105 void set_indices_in_myargstruct_array12(int th,int
n_examples_selected,int j,double delta_w0_value_j,double delta_b56_value) {
6106   int objects_per_worker;
6107   myargstruct_array12[th].j     = j;
6108   myargstruct_array12[th].delta_w0_value_j = delta_w0_value_j;
6109   myargstruct_array12[th].delta_b56_value  = delta_b56_value;
6110   if (n_examples_selected%NUM_THREADS_USED_FOR_WORK==0) {
6111     objects_per_worker = n_examples_selected/NUM_THREADS_USED_FOR_WORK;
6112   } else {
6113     objects_per_worker = n_examples_selected/NUM_THREADS_USED_FOR_WORK +
1;
6114   }
6115   myargstruct_array12[th].loindex = th    * objects_per_worker;
6116   myargstruct_array12[th].hiindex = (th+1)* objects_per_worker - 1;
6117   if (myargstruct_array12[th].hiindex>n_examples_selected-1) {
6118     myargstruct_array12[th].hiindex = n_examples_selected-1;
6119   }
6120 }
6121
6122 __attribute__((optimize("-O3"))) void* worker12(void *p) {
6123   int i;
6124   for (i=((struct myargstruct12*) p)->loindex;i<=((struct
myargstruct12*) p)->hiindex;i++) {
6125     if
(isbitonein_bitposition_in_integerarray(manyruns[i].inputtoprogram,((struct
myargstruct12*) p)->j)) {
6126       manyruns[i].prediction = manyruns[i].prediction + ((struct
myargstruct12*) p)->delta_w0_value_j;
6127     }
6128     manyruns[i].prediction = manyruns[i].prediction + ((struct
myargstruct12*) p)->delta_b56_value;
6129     manyruns[i].error = manyruns[i].prediction - manyruns[i].t;
6130   }
6131   pthread_exit(NULL);
6132 }
6133
6134 void
train_affine_model_update_prediction_and_error_as_part_of_adjust_w0(int
n_examples_selected,int j,double delta_w0_value_j,double delta_b56_value) {
6135   int rc; int th; void* status;
6136   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
6137     pthread_attr_init(&(attrs12[th]));
6138     pthread_attr_setdetachstate(&(attrs12[th]),PTHREAD_CREATE_JOINABLE);
6139
set_indices_in_myargstruct_array12(th,n_examples_selected,j,delta_w0_value_j,
delta_b56_value);
```

```
6140      rc = pthread_create(&(threads12[th]),&(attrs12[th]),worker12,(void*)
(&(myargstruct_array12[th])));
6141      if (rc){
6142        printf("ERROR; return code from pthread_create() is %d\n", rc);
6143        exit(-1);
6144      }
6145    }
6146    for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
6147      rc = pthread_join(threads12[th], &status);
6148      if (rc) {
6149        printf("ERROR; return code from pthread_join() is %d\n", rc);
6150        exit(-1);
6151      }
6152    }
6153    for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
6154      pthread_attr_destroy( &(attrs12[th]) );
6155    }
6156  }
6157
6158  void train_affine_model_adjust_w0(int n_examples_selected) {
6159    int j; double sumerr0; int count0; double sumerr1; int count1; double
delta_w0_value_j; double delta_b56_value;
6160    for (j=0;j<inputsizeinnumberofbits;j++) {
6161
train_affine_model_compute_sumerr_count_for_bit_for_0_and_1(n_examples_select
ed,j,&sumerr0,&count0,&sumerr1,&count1);
6162      if ((count0>0) && (count1>0)) {
6163        delta_w0_value_j = -((sumerr1/count1)-(sumerr0/count0));
6164        delta_b56_value  = -(sumerr0/count0);
6165        p_w0_value[j]    = p_w0_value[j]  + delta_w0_value_j;
6166        p_b56_value[0]   = p_b56_value[0] + delta_b56_value;
6167
train_affine_model_update_prediction_and_error_as_part_of_adjust_w0(n_example
s_selected,j,delta_w0_value_j,delta_b56_value);
6168      } else {
6169        printf("In train_affine_model_adjust_w0. This is not necessarily
an error but it is odd.\n");
6170      }
6171    }
6172  }
6173
6174  void train_affine_model_compute_avgerr(int n_examples_selected,double*
p_avgerr) {
6175    int i; double sum;
6176    sum = manyruns[0].error;
6177    for (i=1;i<n_examples_selected;i++) {
6178      sum = sum + manyruns[0].error;
6179    }
6180  }
6181
6182  void train_affine_model_adjust_b56(int n_examples_selected) {
6183    int i; double avgerr; double delta;
6184    train_affine_model_compute_avgerr(n_examples_selected,&avgerr);
6185    delta = avgerr;
6186    p_b56_value[0] = p_b56_value[0] - delta;
6187    for (i=0;i<n_examples_selected;i++) {
6188      manyruns[i].prediction = manyruns[i].prediction - delta;
```

```
6189      manyruns[i].error = manyruns[i].prediction - manyruns[i].t;
6190    }
6191  }
6192
6193  // void train_affine_model(int n_examples_selected) {
6194  //  train_affine_model_initialize_w0_and_b56(n_examples_selected);
6195  //  train_affine_model_adjust_w0(n_examples_selected);
6196  //  train_affine_model_adjust_b56(n_examples_selected);
6197  // }
6198
6199  void compute_min_prediction_and_max_prediction(int
n_examples_selected,double* p_min_prediction,double* p_max_prediction) {
6200    int i;
6201    (*p_min_prediction) = manyruns[0].prediction;
6202    (*p_max_prediction) = manyruns[0].prediction;
6203    for (i=1;i<=n_examples_selected-1;i++) {
6204      (*p_min_prediction) =
mindouble((*p_min_prediction),manyruns[i].prediction);
6205      (*p_max_prediction) =
maxdouble((*p_max_prediction),manyruns[i].prediction);
6206    }
6207  }
6208
6209  // min_observed_time and max_observed_time are computed by the function
calculate_min_max_average_executiontime_from_manyruns which has been called
earlier
6210  void
adjust_coefficients_of_affine_model_so_that_range_of_predictions_stays_approx
imately_within_range_of_time_measurements(int n_examples_selected) {
6211    double min_prediction; double max_prediction; double factor; int j;
6212
compute_min_prediction_and_max_prediction(n_examples_selected,&min_prediction
,&max_prediction);
6213    factor = (max_prediction-min_prediction)/(max_observed_time-
min_observed_time);
6214    for (j=0;j<inputsizeinnumberofbits;j++) {
6215      p_w0_value[j] = p_w0_value[j] / factor;
6216    }
6217  }
6218
6219  void
adjust_coefficients_of_affine_model_with_basis_functions_so_that_range_of_pre
dictions_stays_approximately_within_range_of_time_measurements(int
n_examples_selected) {
6220    double min_prediction; double max_prediction; double factor; int j;
int basisfunctionindex;
6221
compute_min_prediction_and_max_prediction(n_examples_selected,&min_prediction
,&max_prediction);
6222    factor = (max_prediction-min_prediction)/(max_observed_time-
min_observed_time);
6223    for (j=0;j<inputsizeinnumberofbits;j++) {
6224      p_w0_value[j] = p_w0_value[j] / factor;
6225    }
6226    for
(basisfunctionindex=0;basisfunctionindex<NBASISFUNCTIONS;basisfunctionindex++
) {
```

```
6227     basisfunctions[basisfunctionindex].coefficient =
basisfunctions[basisfunctionindex].coefficient / factor;
6228   }
6229 }
6230
6231 // void generate_basis_functions_without_coefficients() {
6232 //   int iterator; struct basisfunction new_basisfunction;
6233 //   for (iterator=0;iterator<NBASISFUNCTIONS;iterator++) {
6234 //     new_basisfunction = generate_new_basisfunction();
6235 //     while (is_basisfunction_in(iterator,new_basisfunction)) {
6236 //       new_basisfunction = generate_new_basisfunction();
6237 //     }
6238 //     basisfunctions[iterator] = new_basisfunction;
6239 //     basisfunctions[iterator].coefficient = 0.0;
6240 //   }
6241 // }
6242
6243 void
generate_basis_functions_without_coefficients_fill_array_with_contents() {
6244   int iterator; struct basisfunction new_basisfunction;
6245   for (iterator=0;iterator<NBASISFUNCTIONS;iterator++) {
6246     new_basisfunction = generate_new_basisfunction();
6247     basisfunctions[iterator] = new_basisfunction;
6248     basisfunctions[iterator].coefficient = 0.0;
6249   }
6250 }
6251
6252 int array_with_basis_function_indices[NBASISFUNCTIONS];
6253
6254 int
cmpfunc_sort_array_with_basis_function_indices_in_ascending_order(const void
* a, const void * b) {
6255   int* p_a;
6256   int* p_b;
6257   p_a = (int*) a;
6258   p_b = (int*) b;
6259   if (basisfunctions[(*p_a)].bit1_index <
basisfunctions[(*p_b)].bit1_index) {
6260     return -1;
6261   } else {
6262     if (basisfunctions[(*p_a)].bit1_index ==
basisfunctions[(*p_b)].bit1_index) {
6263       if (basisfunctions[(*p_a)].bit2_index <
basisfunctions[(*p_b)].bit2_index) {
6264         return -1;
6265       } else {
6266         if (basisfunctions[(*p_a)].bit2_index ==
basisfunctions[(*p_b)].bit2_index) {
6267           if (basisfunctions[(*p_a)].bit1_value <
basisfunctions[(*p_b)].bit1_value) {
6268             return -1;
6269           } else {
6270             if (basisfunctions[(*p_a)].bit1_value ==
basisfunctions[(*p_b)].bit1_value) {
6271               if (basisfunctions[(*p_a)].bit2_value <
basisfunctions[(*p_b)].bit2_value) {
6272                 return -1;
```

```
6273                } else {
6274                  if (basisfunctions[(*p_a)].bit2_value ==
basisfunctions[(*p_b)].bit2_value) {
6275                    return 0;
6276                  } else {
6277                    return 1;
6278                  }
6279                }
6280              } else {
6281                return 1;
6282              }
6283            }
6284          } else {
6285            return 1;
6286          }
6287        }
6288      } else {
6289        return 1;
6290      }
6291    }
6292  }
6293
6294  void sort_array_with_basis_function_indices_in_ascending_order() {
6295    qsort(array_with_basis_function_indices, NBASISFUNCTIONS, sizeof(int),
cmpfunc_sort_array_with_basis_function_indices_in_ascending_order);
6296  }
6297
6298  void generate_basis_functions_without_coefficients_fill_array_empty() {
6299    int iterator;
6300    for (iterator=0;iterator<NBASISFUNCTIONS;iterator++) {
6301      basisfunctions[iterator].bit1_index = -1;
6302    }
6303  }
6304
6305  void
generate_basis_functions_without_coefficients_fill_array_with_basis_function_
indices() {
6306    int iterator;
6307    for (iterator=0;iterator<NBASISFUNCTIONS;iterator++) {
6308      array_with_basis_function_indices[iterator] = iterator;
6309    }
6310    sort_array_with_basis_function_indices_in_ascending_order();
6311  }
6312
6313  int
generate_basis_functions_without_coefficients_detect_duplicate_and_generate_n
ew() {
6314    int there_was_a_duplicate; int iterator; int
previous_basisfunction_index; int current_basisfunction_index; struct
basisfunction new_basisfunction;
6315    there_was_a_duplicate = 0;
6316    current_basisfunction_index  = array_with_basis_function_indices[0];
6317    for (iterator=1;iterator<NBASISFUNCTIONS;iterator++) {
6318      previous_basisfunction_index = current_basisfunction_index;
6319      current_basisfunction_index  =
array_with_basis_function_indices[iterator];
```

```
6320      while
(basisfunctions_are_identical_quad(basisfunctions[previous_basisfunction_inde
x],basisfunctions[current_basisfunction_index])) {
6321        new_basisfunction = generate_new_basisfunction();
6322        basisfunctions[current_basisfunction_index] = new_basisfunction;
6323        basisfunctions[current_basisfunction_index].coefficient = 0.0;
6324        there_was_a_duplicate = 1;
6325      }
6326    }
6327    return there_was_a_duplicate;
6328  }
6329
6330  void generate_basis_functions_without_coefficients() {
6331    int there_was_a_duplicate;
6332
generate_basis_functions_without_coefficients_fill_array_with_contents();
6333
generate_basis_functions_without_coefficients_fill_array_with_basis_function_
indices();
6334    there_was_a_duplicate =
generate_basis_functions_without_coefficients_detect_duplicate_and_generate_n
ew();
6335    while (there_was_a_duplicate) {
6336
generate_basis_functions_without_coefficients_fill_array_with_basis_function_
indices();
6337      there_was_a_duplicate =
generate_basis_functions_without_coefficients_detect_duplicate_and_generate_n
ew();
6338    }
6339  }
6340
6341  struct myargstruct13 {
6342    struct basisfunction a_basisfunction;
6343    int loindex;
6344    int hiindex;
6345    double sumerr0; int count0; double sumerr1; int count1;
6346  };
6347  pthread_t threads13[NUM_THREADS_USED_FOR_WORK];
6348  pthread_attr_t attrs13[NUM_THREADS_USED_FOR_WORK];
6349  struct myargstruct13 myargstruct_array13[NUM_THREADS_USED_FOR_WORK];
6350
6351  void set_indices_in_myargstruct_array13(int th,int
n_examples_selected,struct basisfunction a_basisfunction) {
6352    int objects_per_worker;
6353    myargstruct_array13[th].a_basisfunction = a_basisfunction;
6354    if (n_examples_selected%NUM_THREADS_USED_FOR_WORK==0) {
6355      objects_per_worker = n_examples_selected/NUM_THREADS_USED_FOR_WORK;
6356    } else {
6357      objects_per_worker = n_examples_selected/NUM_THREADS_USED_FOR_WORK +
1;
6358    }
6359    myargstruct_array13[th].loindex = th    * objects_per_worker;
6360    myargstruct_array13[th].hiindex = (th+1)* objects_per_worker - 1;
6361    if (myargstruct_array13[th].hiindex>n_examples_selected-1) {
6362      myargstruct_array13[th].hiindex = n_examples_selected-1;
6363    }
```

```
6364 }
6365
6366 __attribute__((optimize("-O3"))) void* worker13(void *p) {
6367   int i;
6368   struct basisfunction* p_basisfunction;
6369   p_basisfunction = &(((struct myargstruct13*) p)->a_basisfunction);
6370   ((struct myargstruct13*) p)->sumerr0 = 0.0;
6371   ((struct myargstruct13*) p)->count0  = 0;
6372   ((struct myargstruct13*) p)->sumerr1 = 0.0;
6373   ((struct myargstruct13*) p)->count1  = 0;
6374   for (i=((struct myargstruct13*) p)->loindex;i<=((struct
myargstruct13*) p)->hiindex;i++) {
6375     if (evaluate_basis_on_example_v2_seq(p_basisfunction,i)) {
6376       ((struct myargstruct13*) p)->sumerr1 = ((struct myargstruct13*)
p)->sumerr1 + manyruns[i].error;
6377       ((struct myargstruct13*) p)->count1  = ((struct myargstruct13*)
p)->count1  + 1;
6378     } else {
6379       ((struct myargstruct13*) p)->sumerr0 = ((struct myargstruct13*)
p)->sumerr0 + manyruns[i].error;
6380       ((struct myargstruct13*) p)->count0  = ((struct myargstruct13*)
p)->count0  + 1;
6381     }
6382   }
6383   pthread_exit(NULL);
6384 }
6385
6386 void
train_model_with_basis_functions_compute_sumerr_count_for_bit_for_0_and_1(int
n_examples_selected,struct basisfunction a_basisfunction,double*
p_sumerr0,int* p_count0,double* p_sumerr1,int* p_count1) {
6387   int rc; int th; void* status;
6388   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
6389     pthread_attr_init(&(attrs13[th]));
6390     pthread_attr_setdetachstate(&(attrs13[th]),PTHREAD_CREATE_JOINABLE);
6391
set_indices_in_myargstruct_array13(th,n_examples_selected,a_basisfunction);
6392     rc = pthread_create(&(threads13[th]),&(attrs13[th]),worker13,(void*)
(&(myargstruct_array13[th])));
6393     if (rc){
6394       printf("ERROR; return code from pthread_create() is %d\n", rc);
6395       exit(-1);
6396     }
6397   }
6398   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
6399     rc = pthread_join(threads13[th], &status);
6400     if (rc) {
6401       printf("ERROR; return code from pthread_join() is %d\n", rc);
6402       exit(-1);
6403     }
6404   }
6405   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
6406     pthread_attr_destroy( &(attrs13[th]) );
6407   }
6408   (*p_sumerr0) = myargstruct_array13[0].sumerr0;
6409   (*p_count0)  = myargstruct_array13[0].count0;
6410   (*p_sumerr1) = myargstruct_array13[0].sumerr1;
```

```
6411   (*p_count1)  = myargstruct_array13[0].count1;
6412   for(th=1;th<NUM_THREADS_USED_FOR_WORK;th++) {
6413     (*p_sumerr0) = (*p_sumerr0) + myargstruct_array13[th].sumerr0;
6414     (*p_count0)  = (*p_count0)  + myargstruct_array13[th].count0;
6415     (*p_sumerr1) = (*p_sumerr1) + myargstruct_array13[th].sumerr1;
6416     (*p_count1)  = (*p_count1)  + myargstruct_array13[th].count1;
6417   }
6418 }
6419
6420 struct myargstruct14 {
6421   struct basisfunction a_basisfunction;
6422   double delta_basisfunction_value;
6423   double delta_b56_value;
6424   int loindex;
6425   int hiindex;
6426 };
6427 pthread_t threads14[NUM_THREADS_USED_FOR_WORK];
6428 pthread_attr_t attrs14[NUM_THREADS_USED_FOR_WORK];
6429 struct myargstruct14 myargstruct_array14[NUM_THREADS_USED_FOR_WORK];
6430
6431 void set_indices_in_myargstruct_array14(int th,int
n_examples_selected,struct basisfunction a_basisfunction,double
delta_basisfunction_value,double delta_b56_value) {
6432   int objects_per_worker;
6433   myargstruct_array14[th].a_basisfunction = a_basisfunction;
6434   myargstruct_array14[th].delta_basisfunction_value =
delta_basisfunction_value;
6435   myargstruct_array14[th].delta_b56_value           = delta_b56_value;
6436   if (n_examples_selected%NUM_THREADS_USED_FOR_WORK==0) {
6437     objects_per_worker = n_examples_selected/NUM_THREADS_USED_FOR_WORK;
6438   } else {
6439     objects_per_worker = n_examples_selected/NUM_THREADS_USED_FOR_WORK +
1;
6440   }
6441   myargstruct_array14[th].loindex = th    * objects_per_worker;
6442   myargstruct_array14[th].hiindex = (th+1)* objects_per_worker - 1;
6443   if (myargstruct_array14[th].hiindex>n_examples_selected-1) {
6444     myargstruct_array14[th].hiindex = n_examples_selected-1;
6445   }
6446 }
6447
6448 __attribute__((optimize("-O3"))) void* worker14(void *p) {
6449   int i;
6450   struct basisfunction* p_basisfunction;
6451   p_basisfunction = &(((struct myargstruct14*) p)->a_basisfunction);
6452   for (i=((struct myargstruct14*) p)->loindex;i<=((struct
myargstruct14*) p)->hiindex;i++) {
6453     if (evaluate_basis_on_example_v2_seq(p_basisfunction,i)) {
6454       manyruns[i].prediction = manyruns[i].prediction + ((struct
myargstruct14*) p)->delta_basisfunction_value;
6455     }
6456     manyruns[i].prediction = manyruns[i].prediction + ((struct
myargstruct14*) p)->delta_b56_value;
6457     manyruns[i].error = manyruns[i].prediction - manyruns[i].t;
6458   }
6459   pthread_exit(NULL);
6460 }
```

```
6461
6462 void
train_model_with_basis_functions_update_prediction_and_error_as_part_of_adjus
t_w5(int n_examples_selected,struct basisfunction a_basisfunction,double
delta_basisfunction_value,double delta_b56_value) {
6463   int rc; int th; void* status;
6464   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
6465     pthread_attr_init(&(attrs14[th]));
6466     pthread_attr_setdetachstate(&(attrs14[th]),PTHREAD_CREATE_JOINABLE);
6467
set_indices_in_myargstruct_array14(th,n_examples_selected,a_basisfunction,del
ta_basisfunction_value,delta_b56_value);
6468     rc = pthread_create(&(threads14[th]),&(attrs14[th]),worker14,(void*)
(&(myargstruct_array14[th])));
6469     if (rc){
6470       printf("ERROR; return code from pthread_create() is %d\n", rc);
6471       exit(-1);
6472     }
6473   }
6474   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
6475     rc = pthread_join(threads14[th], &status);
6476     if (rc) {
6477       printf("ERROR; return code from pthread_join() is %d\n", rc);
6478       exit(-1);
6479     }
6480   }
6481   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
6482     pthread_attr_destroy( &(attrs14[th]) );
6483   }
6484 }
6485
6486 void train_model_with_basis_functions_adjust_w5(int n_examples_selected)
{
6487   int basisfunction_iterator; double sumerr0; int count0; double
sumerr1; int count1; double delta_basisfunction_value; double
delta_b56_value;
6488   for
(basisfunction_iterator=0;basisfunction_iterator<NBASISFUNCTIONS;basisfunctio
n_iterator++) {
6489
train_model_with_basis_functions_compute_sumerr_count_for_bit_for_0_and_1(n_e
xamples_selected,basisfunctions[basisfunction_iterator],&sumerr0,&count0,&sum
err1,&count1);
6490     if ((count0>0) && (count1>0)) {
6491       delta_basisfunction_value                                  = -
((sumerr1/count1)-(sumerr0/count0));
6492       delta_b56_value                                            = -
(sumerr0/count0);
6493       basisfunctions[basisfunction_iterator].coefficient =
basisfunctions[basisfunction_iterator].coefficient  +
delta_basisfunction_value;
6494       p_b56_value[0]                                             =
p_b56_value[0]                                    + delta_b56_value;
6495
train_model_with_basis_functions_update_prediction_and_error_as_part_of_adjus
t_w5(n_examples_selected,basisfunctions[basisfunction_iterator],delta_basisfu
nction_value,delta_b56_value);
```

```
6496     } else {
6497       printf("In train_model_with_basis_functions_adjust_w5. This is not
necessarily an error but it is odd.\n");
6498     }
6499   }
6500 }
6501
6502 void find_WCET_input_from_prediction_model_w0_b56_w5(char*
fn_prefix,int* p_int_array,double* p_t) {
6503
find_WCET_input_from_prediction_model_w0_b56_w1_b1_w5(fn_prefix,p_int_array,p
_t);
6504 }
6505
6506 int find_WCET_v3(char* fn_with_executiontimemeasurements_inputs,char*
fn_with_executiontimemeasurements_times,char* fn_prefix,int programid,
6507   int n_examples_in_file, int n_examples_selected) {
6508   char fn[2000]; char fn2[2000]; double t0; double t1; FILE* f;
6509   char temp1_fn_prefix[200];
6510   char temp2_fn_prefix[200];
6511
allocate_memory_for_predictedworstcaseinput_phase0_and_predictedworstcaseinpu
t_phase1();
6512   allocate_memory_for_learnable_weights_v2(); // this allocates memory
for w0 and b56 which we need; we don't need the others that are allocated by
this func
6513   allocate_memory_for_manyruns_v2(n_examples_selected);
6514
fill_many_examples_from_files(fn_with_executiontimemeasurements_inputs,fn_wit
h_executiontimemeasurements_times,n_examples_in_file,n_examples_selected);
6515
calculate_min_max_average_executiontime_from_manyruns(n_examples_selected);
6516   printf("n_examples_in_file = %d\n",n_examples_in_file);
fflush(stdout);
6517   printf("n_examples_selected = %d\n",n_examples_selected);
fflush(stdout);
6518   train_affine_model_initialize_w0_and_b56(n_examples_selected);
6519   train_affine_model_adjust_w0(n_examples_selected);
6520   train_affine_model_adjust_w0(n_examples_selected);
6521
6522   sprintf(temp1_fn_prefix,"%s_for_affine_model_1",fn_prefix);
6523   compute_error_and_prediction_based_on_w0_b56(n_examples_selected);
6524   write_all_predictions_to_file(temp1_fn_prefix,n_examples_selected);
6525   write_all_errors_to_file(temp1_fn_prefix,n_examples_selected);
6526
write_summary_of_predictions_and_errors_to_files(temp1_fn_prefix,n_examples_s
elected);
6527
6528   write_w0_b56_to_textfile(temp1_fn_prefix);
6529   read_w0_b56_from_textfile(temp1_fn_prefix);
6530
find_WCET_input_from_prediction_model_w0_b56(temp1_fn_prefix,predictedworstca
seinput,&predicted_time_predictedworstcaseinput);
6531
copy_inputtoprogram(predictedworstcaseinput_phase0,predictedworstcaseinput);
6532   predicted_time_predictedworstcaseinput_phase0 =
predicted_time_predictedworstcaseinput;
```

```
6533
6534    generate_basis_functions_without_coefficients();
6535    train_model_with_basis_functions_adjust_w5(n_examples_selected);
6536    train_model_with_basis_functions_adjust_w5(n_examples_selected);
6537
6538
sprintf(temp2_fn_prefix,"%s_for_affine_model_with_basis_functions_2",fn_prefi
x);
6539
compute_error_and_prediction_based_on_w0_b56_and_basis_functions(n_examples_s
elected);
6540    write_all_predictions_to_file(temp2_fn_prefix,n_examples_selected);
6541    write_all_errors_to_file(temp2_fn_prefix,n_examples_selected);
6542
write_summary_of_predictions_and_errors_to_files(temp2_fn_prefix,n_examples_s
elected);
6543
6544    print_basis_functions_to_file(temp2_fn_prefix);
6545    read_basis_functions_from_file(temp2_fn_prefix);
6546
find_WCET_input_from_prediction_model_w0_b56_w5(temp2_fn_prefix,predictedwors
tcaseinput,&predicted_time_predictedworstcaseinput);
6547
copy_inputtoprogram(predictedworstcaseinput_phase1,predictedworstcaseinput);
6548    predicted_time_predictedworstcaseinput_phase1 =
predicted_time_predictedworstcaseinput;
6549
6550    printf("Running with obtained worst-case input\n");
6551    sleep(10);
6552    setprocessoraffinity_to_allow_just_a_single_processor_proc0();
6553    sleep(60);
6554
run_program_with_two_different_inputs_dont_use_filesystem_run_X_times(predict
edworstcaseinput_phase0,predictedworstcaseinput_phase1,programid,100,&t0,&t1)
;
6555
6556    sprintf(fn, "%s_0_predictedworstcaseinput.dat",        fn_prefix);
6557    sprintf(fn2,"%s_0_predictedworstcaseexecutiontime.dat",fn_prefix);
6558
write_input_and_execution_time_to_files(fn,fn2,predictedworstcaseinput_phase0
,t0);
6559
sprintf(fn,"%s_0_predictedworstcaseexecutiontime_notrun.dat",fn_prefix); f =
fopen(fn, "w+"); fprintf( f, "%18.12lf\n",
predicted_time_predictedworstcaseinput_phase0); fclose( f);
6560
6561    sprintf(fn, "%s_1_predictedworstcaseinput.dat",        fn_prefix);
6562    sprintf(fn2,"%s_1_predictedworstcaseexecutiontime.dat",fn_prefix);
6563
write_input_and_execution_time_to_files(fn,fn2,predictedworstcaseinput_phase1
,t1);
6564
sprintf(fn,"%s_1_predictedworstcaseexecutiontime_notrun.dat",fn_prefix); f =
fopen(fn, "w+"); fprintf( f, "%18.12lf\n",
predicted_time_predictedworstcaseinput_phase1); fclose( f);
6565
```

```
6566   setprocessoraffinity_to_allow_all_processors(); // we don't need to
restore this but it does not hurt
6567
6568   free_memory_for_manyruns_v2(n_examples_selected);
6569   free_memory_for_learnable_weights_v2(); // we only need w0 and b56
6570
free_memory_for_predictedworstcaseinput_phase0_and_predictedworstcaseinput_ph
ase1();
6571 }
6572
6573 double sumweight;
6574 #define NBUCKETS_USED_FOR_WEIGHING 1000
6575 int buckets_counts[NBUCKETS_USED_FOR_WEIGHING];
6576 double buckets_weights[NBUCKETS_USED_FOR_WEIGHING];
6577 int get_bucket(double v,double minv,double maxv) {
6578   double frac; int b;
6579   frac = (v-minv)/(maxv-minv);
6580   b = frac*NBUCKETS_USED_FOR_WEIGHING;
6581   if (b>=NBUCKETS_USED_FOR_WEIGHING-1) { b = NBUCKETS_USED_FOR_WEIGHING-
1; } // this is to deal with the case that frac = 1
6582   if (b<=0)                            { b = 0;
} // this is to deal with the case that frac = 0 negative
6583   return b;
6584 }
6585 // this assumes that min_observed_time and max_observed_time have been
computed
6586 void compute_buckets_weights_based_on_t(int n_examples_in_file) {
6587   int b; int i; double temp;
6588   for (b=0;b<NBUCKETS_USED_FOR_WEIGHING;b++) { buckets_counts[b] = 0; }
6589   for (i=0;i<n_examples_in_file;i++) {
6590     b = get_bucket(manyruns[i].t,min_observed_time,max_observed_time);
6591     buckets_counts[b] = buckets_counts[b] + 1;
6592   }
6593   temp = 0.0;
6594   for (b=0;b<NBUCKETS_USED_FOR_WEIGHING;b++) {
6595     // if (buckets_counts[b]>0) {
6596     //  temp = temp + 1.0/buckets_counts[b];
6597     // }
6598     temp = temp + 1.0/(1+buckets_counts[b]);
6599   }
6600   for (b=0;b<NBUCKETS_USED_FOR_WEIGHING;b++) {
6601     // if (buckets_counts[b]>0) {
6602     //  buckets_weights[b] = (NBUCKETS_USED_FOR_WEIGHING *
(1.0/buckets_counts[b])) / temp;
6603     // } else {
6604     //  buckets_weights[b] = 0.0;
6605     // }
6606     buckets_weights[b] = (NBUCKETS_USED_FOR_WEIGHING *
(1.0/(1+buckets_counts[b]))) / temp;
6607   }
6608   for (i=0;i<n_examples_in_file;i++) {
6609     b = get_bucket(manyruns[i].t,min_observed_time,max_observed_time);
6610     manyruns[i].weight = buckets_weights[b];
6611   }
6612   sumweight = 0.0;
6613   for (i=0;i<n_examples_in_file;i++) {
6614     sumweight = sumweight + manyruns[i].weight;
```

```
6615   }
6616 }
6617
6618 // void write_buckets_weights_to_file(char* infn,double min_error,double
max_error) {
6619 //   char fn[200]; int b; FILE* f;
6620 //   sprintf(fn,"%s_%18.15lf_%18.15lf.txt",infn,min_error,max_error);
6621 //   f = fopen(fn, "w" ); if (f==NULL) { printf("Error in
write_buckets_weights_to_file\n"); fflush(stdout); exit(-1); }
6622 //   for (b=0;b<NBUCKETS_USED_FOR_WEIGHING;b++) {
6623 //     fprintf(f,"%d %18.15lf\n",buckets_counts[b],buckets_weights[b]);
6624 //   }
6625 //   fclose(f);
6626 // }
6627
6628 void get_min_error_and_max_error(double* p_min_error,double*
p_max_error,int n_examples_in_file) {
6629   int i;
6630   (*p_min_error) = manyruns[0].error;
6631   (*p_max_error) = manyruns[0].error;
6632   for (i=1;i<n_examples_in_file;i++) {
6633     if (manyruns[i].error<(*p_min_error)) { (*p_min_error) =
manyruns[i].error; }
6634     if (manyruns[i].error>(*p_max_error)) { (*p_max_error) =
manyruns[i].error; }
6635   }
6636 }
6637
6638 // here, we need to compute min_err and max_err
6639 void compute_buckets_weights_based_on_err(int n_examples_in_file) {
6640   int b; int i; double temp;
6641   double min_error; double max_error;
6642   get_min_error_and_max_error(&min_error,&max_error,n_examples_in_file);
6643   printf("  min_error = %15.12lf max_error =
%15.12lf\n",min_error,max_error);
6644   for (b=0;b<NBUCKETS_USED_FOR_WEIGHING;b++) { buckets_counts[b] = 0; }
6645   for (i=0;i<n_examples_in_file;i++) {
6646     b = get_bucket(manyruns[i].error,min_error,max_error);
6647     buckets_counts[b] = buckets_counts[b] + 1;
6648   }
6649   temp = 0.0;
6650   for (b=0;b<NBUCKETS_USED_FOR_WEIGHING;b++) {
6651     // if (buckets_counts[b]>0) {
6652     //   temp = temp + 1.0/buckets_counts[b];
6653     // }
6654     temp = temp + 1.0/(1+buckets_counts[b]);
6655   }
6656   for (b=0;b<NBUCKETS_USED_FOR_WEIGHING;b++) {
6657     // if (buckets_counts[b]>0) {
6658     //   buckets_weights[b] = (NBUCKETS_USED_FOR_WEIGHING *
(1.0/buckets_counts[b])) / temp;
6659     // } else {
6660     //   buckets_weights[b] = 0.0;
6661     // }
6662     buckets_weights[b] = (NBUCKETS_USED_FOR_WEIGHING *
(1.0/(1+buckets_counts[b]))) / temp;
6663   }
```

```
6664    //
write_buckets_weights_to_file("temp_buckets_weights",min_error,max_error);
6665    for (i=0;i<n_examples_in_file;i++) {
6666      b = get_bucket(manyruns[i].error,min_error,max_error);
6667      manyruns[i].weight = buckets_weights[b];
6668    }
6669    sumweight = 0.0;
6670    for (i=0;i<n_examples_in_file;i++) {
6671      sumweight = sumweight + manyruns[i].weight;
6672    }
6673  }
6674
6675  void compute_buckets_weights_all_one(int n_examples_in_file) {
6676    int i;
6677    double min_error; double max_error;
6678    get_min_error_and_max_error(&min_error,&max_error,n_examples_in_file);
6679    printf("  min_error = %15.12lf max_error =
%15.12lf\n",min_error,max_error);
6680    for (i=0;i<n_examples_in_file;i++) {
6681      manyruns[i].weight = 1.0;
6682    }
6683    sumweight = 0.0;
6684    for (i=0;i<n_examples_in_file;i++) {
6685      sumweight = sumweight + manyruns[i].weight;
6686    }
6687  }
6688
6689  double compute_avg_execution_time_use_weighting_with_buckets(int
n_examples_selected) {
6690    int i; double sum;
6691    sum = manyruns[0].t * manyruns[0].weight;
6692    for (i=1;i<n_examples_selected;i++) {
6693      sum = sum + manyruns[i].t * manyruns[i].weight;
6694    }
6695    return sum/sumweight;
6696  }
6697
6698  void
train_affine_model_initialize_w0_and_b56_use_weighting_with_buckets(int
n_examples_in_file) {
6699    int i; int j;
6700    compute_buckets_weights_based_on_t(n_examples_in_file);
6701    p_b56_value[0] =
compute_avg_execution_time_use_weighting_with_buckets(n_examples_in_file);
6702    printf("p_b56_value[0] = %lf\n",p_b56_value[0]); fflush(stdout);
6703    for (j=0;j<inputsizeinnumberofbits;j++) {
6704      p_w0_value[j] = 0.0;
6705    }
6706    for (i=0;i<n_examples_in_file;i++) {
6707      manyruns[i].prediction = p_b56_value[0];
6708      manyruns[i].error = manyruns[i].prediction - manyruns[i].t;
6709    }
6710  }
6711
6712  struct myargstruct15 {
6713    int j;
6714    int loindex;
```

```
6715   int hiindex;
6716   double sumerr0; double count0; double sumerr1; double count1;
6717 };
6718 pthread_t threads15[NUM_THREADS_USED_FOR_WORK];
6719 pthread_attr_t attrs15[NUM_THREADS_USED_FOR_WORK];
6720 struct myargstruct15 myargstruct_array15[NUM_THREADS_USED_FOR_WORK];
6721
6722 void set_indices_in_myargstruct_array15(int th,int
n_examples_selected,int j) {
6723   int objects_per_worker;
6724   myargstruct_array15[th].j = j;
6725   if (n_examples_selected%NUM_THREADS_USED_FOR_WORK==0) {
6726     objects_per_worker = n_examples_selected/NUM_THREADS_USED_FOR_WORK;
6727   } else {
6728     objects_per_worker = n_examples_selected/NUM_THREADS_USED_FOR_WORK +
1;
6729   }
6730   myargstruct_array15[th].loindex = th    * objects_per_worker;
6731   myargstruct_array15[th].hiindex = (th+1)* objects_per_worker - 1;
6732   if (myargstruct_array15[th].hiindex>n_examples_selected-1) {
6733     myargstruct_array15[th].hiindex = n_examples_selected-1;
6734   }
6735 }
6736
6737 __attribute__((optimize("-O3"))) void* worker15(void *p) {
6738   int i;
6739   ((struct myargstruct15*) p)->sumerr0 = 0.0;
6740   ((struct myargstruct15*) p)->count0  = 0.0;
6741   ((struct myargstruct15*) p)->sumerr1 = 0.0;
6742   ((struct myargstruct15*) p)->count1  = 0.0;
6743   for (i=((struct myargstruct15*) p)->loindex;i<=((struct
myargstruct15*) p)->hiindex;i++) {
6744     if
(isbitonein_bitposition_in_integerarray(manyruns[i].inputtoprogram,((struct
myargstruct15*) p)->j)) {
6745       ((struct myargstruct15*) p)->sumerr1 = ((struct myargstruct15*)
p)->sumerr1 + manyruns[i].error * manyruns[i].weight;
6746       ((struct myargstruct15*) p)->count1  = ((struct myargstruct15*)
p)->count1  + manyruns[i].weight;
6747     } else {
6748       ((struct myargstruct15*) p)->sumerr0 = ((struct myargstruct15*)
p)->sumerr0 + manyruns[i].error * manyruns[i].weight;
6749       ((struct myargstruct15*) p)->count0  = ((struct myargstruct15*)
p)->count0  + manyruns[i].weight;
6750     }
6751   }
6752   pthread_exit(NULL);
6753 }
6754
6755 void
train_affine_model_compute_sumerr_count_for_bit_for_0_and_1_use_weighting_wit
h_buckets(int n_examples_selected,int j,double* p_sumerr0,double*
p_count0,double* p_sumerr1,double* p_count1) {
6756   int rc; int th; void* status;
6757   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
6758     pthread_attr_init(&(attrs15[th]));
6759     pthread_attr_setdetachstate(&(attrs15[th]),PTHREAD_CREATE_JOINABLE);
```

```
6760      set_indices_in_myargstruct_array15(th,n_examples_selected,j);
6761      rc = pthread_create(&(threads15[th]),&(attrs15[th]),worker15,(void*)
(&(myargstruct_array15[th])));
6762      if (rc){
6763        printf("ERROR; return code from pthread_create() is %d\n", rc);
6764        exit(-1);
6765      }
6766    }
6767    for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
6768      rc = pthread_join(threads15[th], &status);
6769      if (rc) {
6770        printf("ERROR; return code from pthread_join() is %d\n", rc);
6771        exit(-1);
6772      }
6773    }
6774    for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
6775      pthread_attr_destroy( &(attrs15[th]) );
6776    }
6777    (*p_sumerr0) = myargstruct_array15[0].sumerr0;
6778    (*p_count0)  = myargstruct_array15[0].count0;
6779    (*p_sumerr1) = myargstruct_array15[0].sumerr1;
6780    (*p_count1)  = myargstruct_array15[0].count1;
6781    for(th=1;th<NUM_THREADS_USED_FOR_WORK;th++) {
6782      (*p_sumerr0) = (*p_sumerr0) + myargstruct_array15[th].sumerr0;
6783      (*p_count0)  = (*p_count0)  + myargstruct_array15[th].count0;
6784      (*p_sumerr1) = (*p_sumerr1) + myargstruct_array15[th].sumerr1;
6785      (*p_count1)  = (*p_count1)  + myargstruct_array15[th].count1;
6786    }
6787  }
6788
6789  /*
6790  void train_affine_model_adjust_w0_use_weighting_with_buckets(int
n_examples_in_file) {
6791    int j; double sumerr0; double count0; double sumerr1; double count1;
double delta_w0_value_j; double delta_b56_value;
6792    compute_buckets_weights_based_on_err(n_examples_in_file);
6793    for (j=0;j<inputsizeinnumberofbits;j++) {
6794      if (j%1000==0) { printf("j = %d\n",j); fflush(stdout); }
6795
train_affine_model_compute_sumerr_count_for_bit_for_0_and_1_use_weighting_wit
h_buckets(n_examples_in_file,j,&sumerr0,&count0,&sumerr1,&count1);
6796      if ((count0>0.0) && (count1>0.0)) {
6797        delta_w0_value_j = (-((sumerr1/count1)-(sumerr0/count0)))*0.025;
6798        delta_b56_value  = (-(sumerr0/count0))*0.025;
6799        p_w0_value[j]    = p_w0_value[j]  + delta_w0_value_j;
6800        p_b56_value[0]   = p_b56_value[0] + delta_b56_value;
6801
train_affine_model_update_prediction_and_error_as_part_of_adjust_w0(n_example
s_in_file,j,delta_w0_value_j,delta_b56_value);
6802      } else {
6803        printf("In
train_affine_model_adjust_w0_use_weighting_with_buckets. This is not
necessarily an error but it is odd.\n");
6804      }
6805    }
6806  }
6807  */
```

```
6808 void train_affine_model_adjust_w0_use_weighting_with_buckets(int
n_examples_in_file) {
6809   int j; double sumerr0; double count0; double sumerr1; double count1;
double delta_w0_value_j; double delta_b56_value;
6810   compute_buckets_weights_based_on_err(n_examples_in_file);
6811   for (j=0;j<inputsizeinnumberofbits;j++) {
6812     if (j%1000==0) { printf("j = %d\n",j); fflush(stdout); }
6813
train_affine_model_compute_sumerr_count_for_bit_for_0_and_1_use_weighting_wit
h_buckets(n_examples_in_file,j,&sumerr0,&count0,&sumerr1,&count1);
6814     if (count1>0.0) {
6815       if (count0>0.0) {
6816         delta_w0_value_j = (-((sumerr1/count1)-(sumerr0/count0)))*0.025;
6817         delta_b56_value  = (-(sumerr0/count0))*0.025;
6818       } else {
6819         delta_w0_value_j = (-(sumerr1/count1))*0.025;
6820         delta_b56_value  = 0.0;
6821       }
6822       p_w0_value[j]    = p_w0_value[j]  + delta_w0_value_j;
6823       p_b56_value[0]   = p_b56_value[0] + delta_b56_value;
6824
train_affine_model_update_prediction_and_error_as_part_of_adjust_w0(n_example
s_in_file,j,delta_w0_value_j,delta_b56_value);
6825     } else {
6826       printf("In
train_affine_model_adjust_w0_use_weighting_with_buckets. This is not
necessarily an error but it is odd.\n");
6827     }
6828   }
6829 }
6830
6831
6832 struct myargstruct16 {
6833   int j;
6834   int loindex;
6835   int hiindex;
6836   double sumerr0; double count0; double sumerr1; double count1;
6837 };
6838 pthread_t threads16[NUM_THREADS_USED_FOR_WORK];
6839 pthread_attr_t attrs16[NUM_THREADS_USED_FOR_WORK];
6840 struct myargstruct16 myargstruct_array16[NUM_THREADS_USED_FOR_WORK];
6841
6842 void set_indices_in_myargstruct_array16(int th,int
n_examples_selected,int j) {
6843   int objects_per_worker;
6844   myargstruct_array16[th].j = j;
6845   if (n_examples_selected%NUM_THREADS_USED_FOR_WORK==0) {
6846     objects_per_worker = n_examples_selected/NUM_THREADS_USED_FOR_WORK;
6847   } else {
6848     objects_per_worker = n_examples_selected/NUM_THREADS_USED_FOR_WORK +
1;
6849   }
6850   myargstruct_array16[th].loindex = th    * objects_per_worker;
6851   myargstruct_array16[th].hiindex = (th+1)* objects_per_worker - 1;
6852   if (myargstruct_array16[th].hiindex>n_examples_selected-1) {
6853     myargstruct_array16[th].hiindex = n_examples_selected-1;
6854   }
```

```
6855 }
6856
6857 __attribute__((optimize("-O3"))) void* worker16(void *p) {
6858   int i;
6859   ((struct myargstruct16*) p)->sumerr0 = 0.0;
6860   ((struct myargstruct16*) p)->count0  = 0.0;
6861   ((struct myargstruct16*) p)->sumerr1 = 0.0;
6862   ((struct myargstruct16*) p)->count1  = 0.0;
6863   for (i=((struct myargstruct16*) p)->loindex;i<=((struct
myargstruct16*) p)->hiindex;i++) {
6864     if (evaluate_basis_on_programinput(&(basisfunctions[((struct
myargstruct16*) p)->j]),manyruns[i].inputtoprogram)) {
6865       ((struct myargstruct16*) p)->sumerr1 = ((struct myargstruct16*)
p)->sumerr1 + manyruns[i].error * manyruns[i].weight;
6866       ((struct myargstruct16*) p)->count1  = ((struct myargstruct16*)
p)->count1  + manyruns[i].weight;
6867     } else {
6868       ((struct myargstruct16*) p)->sumerr0 = ((struct myargstruct16*)
p)->sumerr0 + manyruns[i].error * manyruns[i].weight;
6869       ((struct myargstruct16*) p)->count0  = ((struct myargstruct16*)
p)->count0  + manyruns[i].weight;
6870     }
6871   }
6872   pthread_exit(NULL);
6873 }
6874
6875 void
train_model_with_basis_functions_compute_sumerr_count_for_bit_for_0_and_1_use
_weighting_with_buckets(int n_examples_selected,int j,double*
p_sumerr0,double* p_count0,double* p_sumerr1,double* p_count1) {
6876   int rc; int th; void* status;
6877   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
6878     pthread_attr_init(&(attrs16[th]));
6879     pthread_attr_setdetachstate(&(attrs16[th]),PTHREAD_CREATE_JOINABLE);
6880     set_indices_in_myargstruct_array16(th,n_examples_selected,j);
6881     rc = pthread_create(&(threads16[th]),&(attrs16[th]),worker16,(void*)
(&(myargstruct_array16[th])));
6882     if (rc){
6883       printf("ERROR; return code from pthread_create() is %d\n", rc);
6884       exit(-1);
6885     }
6886   }
6887   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
6888     rc = pthread_join(threads16[th], &status);
6889     if (rc) {
6890       printf("ERROR; return code from pthread_join() is %d\n", rc);
6891       exit(-1);
6892     }
6893   }
6894   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
6895     pthread_attr_destroy( &(attrs16[th]) );
6896   }
6897   (*p_sumerr0) = myargstruct_array16[0].sumerr0;
6898   (*p_count0)  = myargstruct_array16[0].count0;
6899   (*p_sumerr1) = myargstruct_array16[0].sumerr1;
6900   (*p_count1)  = myargstruct_array16[0].count1;
6901   for(th=1;th<NUM_THREADS_USED_FOR_WORK;th++) {
```

```
6902      (*p_sumerr0) = (*p_sumerr0) + myargstruct_array16[th].sumerr0;
6903      (*p_count0)  = (*p_count0)  + myargstruct_array16[th].count0;
6904      (*p_sumerr1) = (*p_sumerr1) + myargstruct_array16[th].sumerr1;
6905      (*p_count1)  = (*p_count1)  + myargstruct_array16[th].count1;
6906    }
6907  }
6908
6909  struct myargstruct17 {
6910    int j;
6911    int loindex;
6912    int hiindex;
6913    // int delta_w5_value_j;
6914    double delta_w5_value_j;
6915    double delta_b56_value;
6916  };
6917  pthread_t threads17[NUM_THREADS_USED_FOR_WORK];
6918  pthread_attr_t attrs17[NUM_THREADS_USED_FOR_WORK];
6919  struct myargstruct17 myargstruct_array17[NUM_THREADS_USED_FOR_WORK];
6920
6921  void set_indices_in_myargstruct_array17(int th,int
n_examples_selected,int j,double delta_w5_value_j,double delta_b56_value) {
6922    int objects_per_worker;
6923    myargstruct_array17[th].j     = j;
6924    myargstruct_array17[th].delta_w5_value_j = delta_w5_value_j;
6925    myargstruct_array17[th].delta_b56_value  = delta_b56_value;
6926    if (n_examples_selected%NUM_THREADS_USED_FOR_WORK==0) {
6927      objects_per_worker = n_examples_selected/NUM_THREADS_USED_FOR_WORK;
6928    } else {
6929      objects_per_worker = n_examples_selected/NUM_THREADS_USED_FOR_WORK +
1;
6930    }
6931    myargstruct_array17[th].loindex = th    * objects_per_worker;
6932    myargstruct_array17[th].hiindex = (th+1)* objects_per_worker - 1;
6933    if (myargstruct_array17[th].hiindex>n_examples_selected-1) {
6934      myargstruct_array17[th].hiindex = n_examples_selected-1;
6935    }
6936  }
6937
6938  __attribute__((optimize("-O3"))) void* worker17(void *p) {
6939    int i;
6940    for (i=((struct myargstruct17*) p)->loindex;i<=((struct
myargstruct17*) p)->hiindex;i++) {
6941      if (evaluate_basis_on_programinput(&(basisfunctions[((struct
myargstruct17*) p)->j]),manyruns[i].inputtoprogram)) {
6942        manyruns[i].prediction = manyruns[i].prediction + ((struct
myargstruct17*) p)->delta_w5_value_j;
6943      }
6944      manyruns[i].prediction = manyruns[i].prediction + ((struct
myargstruct17*) p)->delta_b56_value;
6945      manyruns[i].error = manyruns[i].prediction - manyruns[i].t;
6946    }
6947    pthread_exit(NULL);
6948  }
6949
6950  void
train_model_with_basis_functions_update_prediction_and_error_as_part_of_adjus
```

```
t_w5_efficient(int n_examples_selected,int j,double delta_w5_value_j,double
delta_b56_value) {
6951   int rc; int th; void* status;
6952   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
6953     pthread_attr_init(&(attrs17[th]));
6954     pthread_attr_setdetachstate(&(attrs17[th]),PTHREAD_CREATE_JOINABLE);
6955
set_indices_in_myargstruct_array17(th,n_examples_selected,j,delta_w5_value_j,
delta_b56_value);
6956     rc = pthread_create(&(threads17[th]),&(attrs17[th]),worker17,(void*)
(&(myargstruct_array17[th])));
6957     if (rc){
6958       printf("ERROR; return code from pthread_create() is %d\n", rc);
6959       exit(-1);
6960     }
6961   }
6962   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
6963     rc = pthread_join(threads17[th], &status);
6964     if (rc) {
6965       printf("ERROR; return code from pthread_join() is %d\n", rc);
6966       exit(-1);
6967     }
6968   }
6969   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
6970     pthread_attr_destroy( &(attrs17[th]) );
6971   }
6972 }
6973
6974 // if we use the learning rate 0.025; if we use 0.025/4, then it seems
like we get convergence for bubblesort with few iterations. But when choosing
the learning rate 0.005, we
6975 // get some stagnation. We tried learning rate 0.001 but this lead to
down-and-up behavior as well.
6976 // Therefore, we use learning rate 0.0003
6977
6978 void
train_model_with_basis_functions_adjust_w5_use_weighting_with_buckets(int
n_examples_in_file) {
6979   int j; double sumerr0; double count0; double sumerr1; double count1;
double delta_w5_value_j; double delta_b56_value;
6980   compute_buckets_weights_based_on_err(n_examples_in_file);
6981   for (j=0;j<NBASISFUNCTIONS;j++) {
6982     if (j%1000==0) { printf("j = %d\n",j); fflush(stdout); }
6983
train_model_with_basis_functions_compute_sumerr_count_for_bit_for_0_and_1_use
_weighting_with_buckets(n_examples_in_file,j,&sumerr0,&count0,&sumerr1,&count
1);
6984     if (count1>0.0) {
6985       if (count0>0.0) {
6986         // delta_w5_value_j = (-((sumerr1/count1)-
(sumerr0/count0)))*0.025/4.0;
6987         // delta_b56_value  = (-(sumerr0/count0))*0.025/4.0;
6988         // delta_w5_value_j = (-((sumerr1/count1)-
(sumerr0/count0)))*0.005;
6989         // delta_b56_value  = (-(sumerr0/count0))*0.005;
6990         // delta_w5_value_j = (-((sumerr1/count1)-
(sumerr0/count0)))*0.001;
```

```
6991          // delta_b56_value  = (-(sumerr0/count0))*0.001;
6992          delta_w5_value_j = (-((sumerr1/count1)-
(sumerr0/count0)))*0.0003;
6993          delta_b56_value  = (-(sumerr0/count0))*0.0003;
6994        } else {
6995          // delta_w5_value_j = (-(sumerr1/count1))*0.025/4.0;
6996          // delta_w5_value_j = (-(sumerr1/count1))*0.005;
6997          // delta_b56_value  = 0.0;
6998          // delta_w5_value_j = (-(sumerr1/count1))*0.001;
6999          delta_w5_value_j = (-(sumerr1/count1))*0.0003;
7000          delta_b56_value  = 0.0;
7001          printf("In
train_affine_model_adjust_w0_use_weighting_with_buckets. This is not
necessarily an error but it is odd. count0==0\n");
7002        }
7003        basisfunctions[j].coefficient = basisfunctions[j].coefficient +
delta_w5_value_j;
7004        p_b56_value[0]   = p_b56_value[0] + delta_b56_value;
7005
train_model_with_basis_functions_update_prediction_and_error_as_part_of_adjus
t_w5_efficient(n_examples_in_file,j,delta_w5_value_j,delta_b56_value);
7006      } else {
7007        printf("In
train_affine_model_adjust_w0_use_weighting_with_buckets. This is not
necessarily an error but it is odd.\n");
7008      }
7009    }
7010 }
7011
7012 struct myargstruct_horizontal_scan {
7013   int loindex;
7014   int hiindex;
7015   double derivative_of_loss_wrt_j[inputsizeinnumberofbits];
7016   double derivative_of_loss_wrt_Nminusone;
7017   double derivative_of_loss_wrt_k[NBASISFUNCTIONS];
7018 };
7019 pthread_t threads_horizontal_scan[NUM_THREADS_USED_FOR_WORK];
7020 pthread_attr_t attrs_horizontal_scan[NUM_THREADS_USED_FOR_WORK];
7021 struct myargstruct_horizontal_scan
myargstruct_array_horizontal_scan[NUM_THREADS_USED_FOR_WORK];
7022
7023 double derivative_of_loss_wrt_j[inputsizeinnumberofbits];
7024 double derivative_of_loss_wrt_Nminusone;
7025 double derivative_of_loss_wrt_k[NBASISFUNCTIONS];
7026
7027 void set_indices_in_myargstruct_array_horizontal_scan(int th,int
n_examples_in_file) {
7028   int objects_per_worker; int j; int k;
7029   for (j=0;j<inputsizeinnumberofbits;j++) {
myargstruct_array_horizontal_scan[th].derivative_of_loss_wrt_j[j] = 0.0; }
7030   myargstruct_array_horizontal_scan[th].derivative_of_loss_wrt_Nminusone
= 0.0;
7031   for (k=0;k<NBASISFUNCTIONS;k++)          {
myargstruct_array_horizontal_scan[th].derivative_of_loss_wrt_k[k] = 0.0; }
7032   if (n_examples_in_file%NUM_THREADS_USED_FOR_WORK==0) {
7033     objects_per_worker = n_examples_in_file/NUM_THREADS_USED_FOR_WORK;
7034   } else {
```

```
7035      objects_per_worker = n_examples_in_file/NUM_THREADS_USED_FOR_WORK +
1;
7036    }
7037    myargstruct_array_horizontal_scan[th].loindex = th    *
objects_per_worker;
7038    myargstruct_array_horizontal_scan[th].hiindex = (th+1)*
objects_per_worker - 1;
7039    if (myargstruct_array_horizontal_scan[th].hiindex>n_examples_in_file-
1) {
7040      myargstruct_array_horizontal_scan[th].hiindex = n_examples_in_file-
1;
7041    }
7042  }
7043
7044  __attribute__((optimize("-O3"))) void* worker_horizontal_scan(void *p) {
7045    int i; int j; int k;
7046    struct myargstruct_horizontal_scan* p2;
7047    p2 = (struct myargstruct_horizontal_scan*) p;
7048    /*
7049    for (i=p2->loindex;i<=p2->hiindex;i++) {
7050      for (j=0;j<inputsizeinnumberofbits;j++) {
7051        p2->derivative_of_loss_wrt_j[j] +=
(manyruns[i].weight*(2*manyruns[i].error*isbitonein_bitposition_in_integerarr
ay(manyruns[i].inputtoprogram,j)));
7052      }
7053    }
7054    for (i=p2->loindex;i<=p2->hiindex;i++) {
7055      p2->derivative_of_loss_wrt_Nminusone +=
(manyruns[i].weight*(2*manyruns[i].error*1.0));
7056    }
7057    */
7058    for (i=p2->loindex;i<=p2->hiindex;i++) {
7059      for (k=0;k<NBASISFUNCTIONS;k++) {
7060        p2->derivative_of_loss_wrt_k[k] +=
(manyruns[i].weight*(2*manyruns[i].error*evaluate_basis_on_programinput(&(bas
isfunctions[k]),manyruns[i].inputtoprogram)));
7061      }
7062    }
7063    pthread_exit(NULL);
7064  }
7065
7066  // not average abs error
7067  double compute_average_error_with_weight(int n_examples_in_file) {
7068    int i; double sum_error; double res;
7069    sum_error = 0.0;
7070    for (i=0;i<n_examples_in_file;i++) {
7071      sum_error = sum_error + manyruns[i].weight*manyruns[i].error;
7072    }
7073    sum_error /= sumweight;
7074    res = sum_error;
7075    return res;
7076  }
7077
7078  // not average abs error
7079  double compute_average_error_without_weight(int n_examples_in_file) {
7080    int i; double sum_error; double res;
7081    sum_error = 0.0;
```

```
7082   for (i=0;i<n_examples_in_file;i++) {
7083     sum_error = sum_error + manyruns[i].error;
7084   }
7085   sum_error /= n_examples_in_file;
7086   res = sum_error;
7087   return res;
7088 }
7089
7090
7091 double sqr_double(double t) {
7092   return t*t;
7093 }
7094
7095 double compute_rms_error(int n_examples_in_file) {
7096   int i; double sum_square_error; double res;
7097   sum_square_error = 0.0;
7098   for (i=0;i<n_examples_in_file;i++) {
7099     sum_square_error = sum_square_error +
manyruns[i].weight*sqr_double(manyruns[i].error);
7100   }
7101   // printf("  In compute_rms_error: sum_square_error = %21.18lf\n",
sum_square_error );
7102   sum_square_error /= sumweight;
7103   // printf("  In compute_rms_error: sum_square_error = %21.18lf\n",
sum_square_error );
7104   res = sqrt(sum_square_error);
7105   // printf("  In compute_rms_error: res = %21.18lf\n", res );
7106   return res;
7107 }
7108
7109 void
train_model_with_basis_functions_adjust_w5_use_weighting_with_buckets_horizon
tal_scan(int n_examples_in_file,double alpha) {
7110   int rc; int th; void* status; int j; int k;
7111   // double alpha;
7112   double rmserror;
7113   compute_buckets_weights_based_on_err(n_examples_in_file);
7114   // compute_buckets_weights_all_one(n_examples_in_file);
7115   // averageerror = compute_average_error(n_examples_in_file); // not
average abs error
7116   rmserror = compute_rms_error(n_examples_in_file);
7117
7118   // printf("\n");
7119   printf("  rmserror = %15.12lf\n", rmserror );
7120   printf("  sumweight = %15.12lf\n", sumweight );
7121   // printf("\n");
7122
7123   for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
7124     pthread_attr_init(&(attrs_horizontal_scan[th]));
7125
pthread_attr_setdetachstate(&(attrs_horizontal_scan[th]),PTHREAD_CREATE_JOINA
BLE);
7126
set_indices_in_myargstruct_array_horizontal_scan(th,n_examples_in_file);
7127     rc =
pthread_create(&(threads_horizontal_scan[th]),&(attrs_horizontal_scan[th]),wo
rker_horizontal_scan,(void*) (&(myargstruct_array_horizontal_scan[th])));
```

```
7128      if (rc){
7129        printf("ERROR; return code from pthread_create() is %d\n", rc);
7130        exit(-1);
7131      }
7132    }
7133    for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
7134      rc = pthread_join(threads_horizontal_scan[th], &status);
7135      if (rc) {
7136        printf("ERROR; return code from pthread_join() is %d\n", rc);
7137        exit(-1);
7138      }
7139    }
7140    for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
7141      pthread_attr_destroy( &(attrs_horizontal_scan[th]) );
7142    }
7143
7144    // for (j=0;j<inputsizeinnumberofbits;j++) {
derivative_of_loss_wrt_j[j] = 0.0; }
7145    // derivative_of_loss_wrt_Nminusone = 0.0;
7146    for (k=0;k<NBASISFUNCTIONS;k++)          { derivative_of_loss_wrt_k[k]
= 0.0; }
7147    for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
7148      // for (j=0;j<inputsizeinnumberofbits;j++) {
derivative_of_loss_wrt_j[j] +=
(myargstruct_array_horizontal_scan[th].derivative_of_loss_wrt_j[j]); }
7149      // derivative_of_loss_wrt_Nminusone
+= (myargstruct_array_horizontal_scan[th].derivative_of_loss_wrt_Nminusone);
7150      for (k=0;k<NBASISFUNCTIONS;k++)          {
derivative_of_loss_wrt_k[k] +=
(myargstruct_array_horizontal_scan[th].derivative_of_loss_wrt_k[k]); }
7151    }
7152    // for (j=0;j<inputsizeinnumberofbits;j++) {
derivative_of_loss_wrt_j[j] /= sumweight; }
7153    // derivative_of_loss_wrt_Nminusone /= sumweight;
7154    for (k=0;k<NBASISFUNCTIONS;k++)          { derivative_of_loss_wrt_k[k]
/= sumweight; }
7155
7156    // for (j=0;j<inputsizeinnumberofbits;j++) {
derivative_of_loss_wrt_j[j] *= (0.5/rmserror); }
7157    // derivative_of_loss_wrt_Nminusone *= (0.5/rmserror);
7158    for (k=0;k<NBASISFUNCTIONS;k++)          { derivative_of_loss_wrt_k[k]
*= (0.5/rmserror); }
7159
7160    /*
7161    printf("\n");
7162    printf("
myargstruct_array_horizontal_scan[0].derivative_of_loss_wrt_j[0] =
%15.12lf\n", myargstruct_array_horizontal_scan[0].derivative_of_loss_wrt_j[0]
);
7163    printf("
myargstruct_array_horizontal_scan[0].derivative_of_loss_wrt_j[1] =
%15.12lf\n", myargstruct_array_horizontal_scan[0].derivative_of_loss_wrt_j[1]
);
7164    printf("
myargstruct_array_horizontal_scan[1].derivative_of_loss_wrt_j[0] =
%15.12lf\n", myargstruct_array_horizontal_scan[1].derivative_of_loss_wrt_j[0]
);
```

```
7165   printf("
myargstruct_array_horizontal_scan[1].derivative_of_loss_wrt_j[1] =
%15.12lf\n", myargstruct_array_horizontal_scan[1].derivative_of_loss_wrt_j[1]
);
7166   printf("  derivative_of_loss_wrt_j[0] = %15.12lf\n",
derivative_of_loss_wrt_j[0]      );
7167   printf("  derivative_of_loss_wrt_j[1] = %15.12lf\n",
derivative_of_loss_wrt_j[1]      );
7168   printf("  derivative_of_loss_wrt_j[32766] = %15.12lf\n",
derivative_of_loss_wrt_j[32766]  );
7169   printf("  derivative_of_loss_wrt_j[32767] = %15.12lf\n",
derivative_of_loss_wrt_j[32767]  );
7170   printf("  derivative_of_loss_wrt_Nminusone =
%15.12lf\n",derivative_of_loss_wrt_Nminusone );
7171   printf("  derivative_of_loss_wrt_k[0] = %15.12lf\n",
derivative_of_loss_wrt_k[0]      );
7172   printf("  derivative_of_loss_wrt_k[1] = %15.12lf\n",
derivative_of_loss_wrt_k[1]      );
7173   printf("  derivative_of_loss_wrt_k[2] = %15.12lf\n",
derivative_of_loss_wrt_k[2]      );
7174   printf("  derivative_of_loss_wrt_k[3] = %15.12lf\n",
derivative_of_loss_wrt_k[3]      );
7175
7176   printf("  p_w0_value[0] = %15.12lf\n",p_w0_value[0]);
7177   printf("  p_w0_value[1] = %15.12lf\n",p_w0_value[1]);
7178   printf("  p_w0_value[32766] = %15.12lf\n",p_w0_value[32766]);
7179   printf("  p_w0_value[32767] = %15.12lf\n",p_w0_value[32767]);
7180   printf("  (*p_b56_value) = %15.12lf\n",(*p_b56_value));
7181   printf("  basisfunctions[0].coefficient =
%15.12lf\n",basisfunctions[0].coefficient);
7182   printf("  basisfunctions[1].coefficient =
%15.12lf\n",basisfunctions[1].coefficient);
7183   printf("  basisfunctions[2].coefficient =
%15.12lf\n",basisfunctions[2].coefficient);
7184   printf("  basisfunctions[3].coefficient =
%15.12lf\n",basisfunctions[3].coefficient);
7185   */
7186
7187   // alpha = 0.001;
7188
7189   /*
7190   for (j=0;j<inputsizeinnumberofbits;j++) {
7191     p_w0_value[j] = p_w0_value[j] - alpha * derivative_of_loss_wrt_j[j];
7192   }
7193   (*p_b56_value) = (*p_b56_value) - alpha *
derivative_of_loss_wrt_Nminusone;
7194   */
7195   for (k=0;k<NBASISFUNCTIONS;k++) {
7196     basisfunctions[k].coefficient = basisfunctions[k].coefficient -
alpha * derivative_of_loss_wrt_k[k];
7197   }
7198
7199   // print_derivative_of_loss_wrt_k_to_file();
7200   // print_basisfunctions_coefficient();
7201
7202   /*
7203   printf("  p_w0_value[0] = %15.12lf\n",p_w0_value[0]);
```

```
7204   printf("  p_w0_value[1] = %15.12lf\n",p_w0_value[1]);
7205   printf("  p_w0_value[32766] = %15.12lf\n",p_w0_value[32766]);
7206   printf("  p_w0_value[32767] = %15.12lf\n",p_w0_value[32767]);
7207   printf("  (*p_b56_value) = %15.12lf\n",(*p_b56_value));
7208   printf("  basisfunctions[0].coefficient =
%15.12lf\n",basisfunctions[0].coefficient);
7209   printf("  basisfunctions[1].coefficient =
%15.12lf\n",basisfunctions[1].coefficient);
7210   printf("  basisfunctions[2].coefficient =
%15.12lf\n",basisfunctions[2].coefficient);
7211   printf("  basisfunctions[3].coefficient =
%15.12lf\n",basisfunctions[3].coefficient);
7212   printf("\n");
7213   */
7214 }
7215
7216 // REASONING: Let N denote the number of bits. In the code it is stored
as inputsizeinnumberofbits
7217 //            Given row,col where row in {0..N-1} and col in {0..N-1},
we can get the index of the array as follows:
7218 //              (sum_{k in {0..row-1}} (N-1-k)) + (col-(row+1))
7219 //            Rewriting yields:
7220 //              (2*N-row-1)*row/2 + col-row-1
7221 // int get_index_to_basisfunctions_to_explore_from_row_and_col(int
row,int col) {
7222 //   return (2*inputsizeinnumberofbits-row-1)*row/2 + col-row-1;
7223 // }
7224 long long get_index_to_basisfunctions_to_explore_from_row_and_col(int
row,int col) {
7225   long long temp1; long long temp2; long long temp3; long long
ret_value;
7226   temp1 = 2*inputsizeinnumberofbits-row-1;
7227   temp2 = row;
7228   temp1 = temp1*temp2;
7229   temp1 = temp1/2;
7230   temp2 = col-row-1;
7231   ret_value = temp1 + temp2;
7232   return ret_value;
7233 }
7234
7235 void get_row_and_col_from_index_to_basisfunctions_to_explore(long long
idx,int *p_row, int *p_col) {
7236   double temp1; double temp2; double row_double; int row; int col; long
long temp_idx;
7237   // Consider (2*N-row-1)*row/2 + col-row-1 = idx
7238   // Let us set col = row+1. This yields:
7239   //  (2*N-row-1)*row/2 + (row+1)-row-1 = idx
7240   // Rewriting yields:
7241   //  (2*N-row-1)*row/2                 = idx
7242   // Multiplying both sides by (-2) and rewriting yields:
7243   //  (row-(2*N-1))*row                 = -2*idx
7244   // Rewriting yields:
7245   //  (row-2*(N-1/2))*row               = -2*idx
7246   // Rewriting yields:
7247   //  (row-(N-1/2))^2 - (N-1/2)^2       = -2*idx
7248   // Rewriting yields:
7249   //  (row-(N-1/2))^2 = (N-1/2)^2         -2*idx
```

```
7250   // Rewriting yields:
7251   //   row-(N-1/2)    = +-sqrt( (N-1/2)^2 -2*idx )
7252   // Rewriting yields:
7253   //   row = (N-1/2) +-sqrt( (N-1/2)^2    -2*idx )
7254   // We know that row<=N-1. Hence, we should choose the subtraction.
Hence:
7255   //   row = (N-1/2)  -sqrt( (N-1/2)^2    -2*idx )
7256   temp1 = (inputsizeinnumberofbits-0.5);
7257   temp2 = sqrt(sqrdouble(temp1)-2*idx);
7258   row_double = temp1 - temp2;
7259   row = row_double;
7260   col = row+1;
7261   temp_idx =
get_index_to_basisfunctions_to_explore_from_row_and_col(row,col);
7262   col = col + idx-temp_idx;
7263   *p_row = row;
7264   *p_col = col;
7265 }
7266
7267 void testing_get_row_and_col_routines() {
7268   int row; int col; long long idx; int ret_row; int ret_col; int OK;
7269   for (row=0;row<32768;row++) {
7270     for (col=row+1;col<=32767;col++) {
7271       idx =
get_index_to_basisfunctions_to_explore_from_row_and_col(row,col);
7272
get_row_and_col_from_index_to_basisfunctions_to_explore(idx,&ret_row,&ret_col
);
7273       OK = (row==ret_row) && (col==ret_col);
7274       if (!OK) {
7275         printf("Error: row = %d col = %d idx = %lld\n",row,col,idx);
7276         exit(-1);
7277       }
7278     }
7279   }
7280   exit(-1);
7281 }
7282
7283 struct onerun* manyruns_for_finding_basis_functions;
7284
7285 void allocate_memory_for_runs_used_for_finding_basis_functions(int
nexamples_for_finding_basis_functions) {
7286   int i;
7287   manyruns_for_finding_basis_functions = (struct onerun*) malloc(
sizeof(struct onerun) * nexamples_for_finding_basis_functions);
7288   if (manyruns_for_finding_basis_functions==NULL) { printf("Error in
allocate_memory_for_runs_used_for_finding_basis_functions. Failed when
allocating manyruns_for_finding_basis_functions.\n"); exit(-1); }
7289   for (i=0;i<nexamples_for_finding_basis_functions;i++) {
7290     manyruns_for_finding_basis_functions[i].inputtoprogram = (int*)
malloc(getinputtoprogram_size_in_number_of_bytes());
7291     if (manyruns_for_finding_basis_functions[i].inputtoprogram==NULL) {
printf("Error in allocate_memory_for_runs_used_for_finding_basis_functions.
Failed when allocating
manyruns_for_finding_basis_functions[i].inputtoprogram.\n"); exit(-1); }
7292   }
7293 }
```

```
7294
7295 void free_memory_for_runs_used_for_finding_basis_functions(int
nexamples_for_finding_basis_functions) {
7296   int i;
7297   for (i=0;i<nexamples_for_finding_basis_functions;i++) {
7298     free(manyruns_for_finding_basis_functions[i].inputtoprogram);
7299   }
7300   free(manyruns_for_finding_basis_functions);
7301 }
7302
7303 int* array_with_indices_for_sorting_descending_abserror;
7304 int
cmpfunc_sort_array_with_indices_for_sorting_descending_abserror(const void *
a, const void * b) {
7305   int* p_a; int* p_b; double abs_err_a; double abs_err_b;
7306   p_a = (int*) a;
7307   p_b = (int*) b;
7308   // abs_err_a =
fabs(manyruns[array_with_indices_for_sorting_descending_abserror[(*p_a)]].err
or);
7309   // abs_err_b =
fabs(manyruns[array_with_indices_for_sorting_descending_abserror[(*p_b)]].err
or);
7310   abs_err_a = fabs(manyruns[(*p_a)].error);
7311   abs_err_b = fabs(manyruns[(*p_b)].error);
7312   if (abs_err_a>abs_err_b) {
7313     return -1;
7314   } else {
7315     if (abs_err_a==abs_err_b) {
7316       return 0;
7317     } else {
7318       return 1;
7319     }
7320   }
7321 }
7322
7323 void fill_memory_for_runs_used_for_finding_basis_functions(int
nexamples_for_finding_basis_functions,int n_examples_in_file) {
7324   int iterator; long long it2;
7325   array_with_indices_for_sorting_descending_abserror =
malloc(n_examples_in_file*sizeof(int)); if
(array_with_indices_for_sorting_descending_abserror==NULL) { printf("Error in
fill_memory_for_runs_used_for_finding_basis_functions\n"); exit(-1); }
7326   for (iterator=0;iterator<n_examples_in_file;iterator++) {
7327     array_with_indices_for_sorting_descending_abserror[iterator] =
iterator;
7328   }
7329
qsort(array_with_indices_for_sorting_descending_abserror,n_examples_in_file,s
izeof(int),cmpfunc_sort_array_with_indices_for_sorting_descending_abserror);
7330   for
(iterator=0;iterator<nexamples_for_finding_basis_functions;iterator++) {
7331     it2 = array_with_indices_for_sorting_descending_abserror[iterator];
7332     copy_inputtoprogram(
manyruns_for_finding_basis_functions[iterator].inputtoprogram,
manyruns[it2].inputtoprogram);
```

```
7333      manyruns_for_finding_basis_functions[iterator].error =
manyruns[it2].error;
7334    }
7335    free(array_with_indices_for_sorting_descending_abserror);
7336  }
7337
7338  long long n_pairs;
7339  struct basisfunction* basisfunctions_to_explore;
7340
7341  void fill_basisfunctions_to_explore_sequential() {
7342    int j1; int j2; int v1; int v2; long long temp_idx;
7343    for (j1=0;j1<inputsizeinnumberofbits-1;j1++) {
7344      for (j2=j1+1;j2<inputsizeinnumberofbits;j2++) {
7345        for (v1=0;v1<=1;v1++) {
7346          for (v2=0;v2<=1;v2++) {
7347            temp_idx =
get_index_to_basisfunctions_to_explore_from_row_and_col(j1,j2);
7348            temp_idx = temp_idx*4+v1*2+v2;
7349            basisfunctions_to_explore[temp_idx].bit1_index = j1;
7350            basisfunctions_to_explore[temp_idx].bit2_index = j2;
7351            basisfunctions_to_explore[temp_idx].bit1_value = v1;
7352            basisfunctions_to_explore[temp_idx].bit2_value = v2;
7353          }
7354        }
7355      }
7356    }
7357  }
7358
7359  // #define NEXAMPLES_FOR_FINDING_BASISFUNCTIONS 100
7360  // #define NEXAMPLES_FOR_FINDING_BASISFUNCTIONS 10
7361  // #define NEXAMPLES_FOR_FINDING_BASISFUNCTIONS 100
7362  // #define NEXAMPLES_FOR_FINDING_BASISFUNCTIONS 10
7363  // #define NEXAMPLES_FOR_FINDING_BASISFUNCTIONS 1
7364  // #define NEXAMPLES_FOR_FINDING_BASISFUNCTIONS 10
7365  // #define NEXAMPLES_FOR_FINDING_BASISFUNCTIONS 100
7366  // #define NEXAMPLES_FOR_FINDING_BASISFUNCTIONS 10000
7367  // #define NEXAMPLES_FOR_FINDING_BASISFUNCTIONS 100
7368  // #define NEXAMPLES_FOR_FINDING_BASISFUNCTIONS 30000
7369  // #define NEXAMPLES_FOR_FINDING_BASISFUNCTIONS 10000
7370  // #define NEXAMPLES_FOR_FINDING_BASISFUNCTIONS 100
7371  // #define NEXAMPLES_FOR_FINDING_BASISFUNCTIONS 1000
7372  // #define NEXAMPLES_FOR_FINDING_BASISFUNCTIONS 10000
7373  // #define NEXAMPLES_FOR_FINDING_BASISFUNCTIONS 10
7374  // #define NEXAMPLES_FOR_FINDING_BASISFUNCTIONS 10000
7375  // #define NEXAMPLES_FOR_FINDING_BASISFUNCTIONS 30000
7376  // #define NEXAMPLES_FOR_FINDING_BASISFUNCTIONS 100000
7377  // #define NEXAMPLES_FOR_FINDING_BASISFUNCTIONS 1000
7378  // #define NEXAMPLES_FOR_FINDING_BASISFUNCTIONS 10000
7379  // #define NEXAMPLES_FOR_FINDING_BASISFUNCTIONS 30000
7380  // #define NEXAMPLES_FOR_FINDING_BASISFUNCTIONS 1000
7381  // #define NEXAMPLES_FOR_FINDING_BASISFUNCTIONS 10000
7382  #define NEXAMPLES_FOR_FINDING_BASISFUNCTIONS 30000
7383
7384  struct myargstruct18 {
7385    long long loindex;
7386    long long hiindex;
7387  };
```

```
7388 pthread_t threads18[NUM_THREADS_USED_FOR_WORK];
7389 pthread_attr_t attrs18[NUM_THREADS_USED_FOR_WORK];
7390 struct myargstruct18 myargstruct_array18[NUM_THREADS_USED_FOR_WORK];
7391
7392 void set_indices_in_myargstruct_array18(int th) {
7393   int objects_per_worker;
7394   if (n_pairs%NUM_THREADS_USED_FOR_WORK==0) {
7395     objects_per_worker = n_pairs/NUM_THREADS_USED_FOR_WORK;
7396   } else {
7397     objects_per_worker = n_pairs/NUM_THREADS_USED_FOR_WORK + 1;
7398   }
7399   myargstruct_array18[th].loindex = th    * objects_per_worker;
7400   myargstruct_array18[th].hiindex = (th+1)* objects_per_worker - 1;
7401   if (myargstruct_array18[th].hiindex>n_pairs-1) {
7402     myargstruct_array18[th].hiindex = n_pairs-1;
7403   }
7404 }
7405
7406 /*
7407 void set_score(long long temp_idx) {
7408   int count; int i;
7409   count = 0;
7410   for (i=0;i<NEXAMPLES_FOR_FINDING_BASISFUNCTIONS;i++) {
7411     if
(evaluate_basis_on_programinput(&(basisfunctions_to_explore[temp_idx]),manyru
ns_for_finding_basis_functions[i].inputtoprogram)) {
7412       count = count + 1;
7413     }
7414   }
7415   basisfunctions_to_explore[temp_idx].score = count;
7416 }
7417 */
7418
7419 void set_score(long long temp_idx) {
7420   double sumerr1; int count1; double sumerr0; int count0; int i; double
delta;
7421   sumerr1 = 0.0; count1 = 0; sumerr0 = 0.0; count0 = 0;
7422   for (i=0;i<NEXAMPLES_FOR_FINDING_BASISFUNCTIONS;i++) {
7423     if
(evaluate_basis_on_programinput(&(basisfunctions_to_explore[temp_idx]),manyru
ns_for_finding_basis_functions[i].inputtoprogram)) {
7424       sumerr1 = sumerr1 + manyruns_for_finding_basis_functions[i].error;
7425       count1 = count1 + 1;
7426     } else {
7427       sumerr0 = sumerr0 + manyruns_for_finding_basis_functions[i].error;
7428       count0 = count0 + 1;
7429     }
7430   }
7431   if (count1>0) {
7432     if (count0>0) {
7433       delta = (-((sumerr1/count1)-(sumerr0/count0)));
7434     } else {
7435       delta = (-(sumerr1/count1));
7436     }
7437     // basisfunctions_to_explore[temp_idx].score = fabs(delta);
7438     basisfunctions_to_explore[temp_idx].delta = delta;
7439   } else {
```

```
7440      // basisfunctions_to_explore[temp_idx].score = 0.0;
7441      basisfunctions_to_explore[temp_idx].delta = 0.0;
7442    }
7443    basisfunctions_to_explore[temp_idx].score = fabs(delta);
7444  }
7445
7446  __attribute__((optimize("-O3"))) void* worker18(void *p) {
7447    int temp_idx;
7448    for (temp_idx=((struct myargstruct18*) p)->loindex;temp_idx<=((struct
myargstruct18*) p)->hiindex;temp_idx++) {
7449      set_score(temp_idx);
7450    }
7451    pthread_exit(NULL);
7452  }
7453
7454  void evaluate_basisfunctions_to_explore_parallel() {
7455    int rc; int th; void* status;
7456    for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
7457      pthread_attr_init(&(attrs18[th]));
7458      pthread_attr_setdetachstate(&(attrs18[th]),PTHREAD_CREATE_JOINABLE);
7459      set_indices_in_myargstruct_array18(th);
7460      rc = pthread_create(&(threads18[th]),&(attrs18[th]),worker18,(void*)
(&(myargstruct_array18[th])));
7461      if (rc){
7462        printf("ERROR; return code from pthread_create() is %d\n", rc);
7463        exit(-1);
7464      }
7465    }
7466    for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
7467      rc = pthread_join(threads18[th], &status);
7468      if (rc) {
7469        printf("ERROR; return code from pthread_join() is %d\n", rc);
7470        exit(-1);
7471      }
7472    }
7473    for(th=0;th<NUM_THREADS_USED_FOR_WORK;th++) {
7474      pthread_attr_destroy( &(attrs18[th]) );
7475    }
7476  }
7477
7478  void read_basisfunctions_to_explore_from_file(char* fn_prefix) {
7479    FILE* f; char fn[200]; int temp_idx;
7480    sprintf(fn,"%s_basisfunctions_to_explore.dat",fn_prefix);
7481    f = fopen(fn, "rb");
7482    for (temp_idx=0;temp_idx<n_pairs;temp_idx++) {
7483      fread(&(basisfunctions_to_explore[temp_idx]),sizeof(struct
basisfunction),1,f);
7484    }
7485    fclose( f);
7486  }
7487
7488  void read_basisfunctions_to_explore_from_file_temp(char* fn_prefix) {
7489    FILE* f; char fn[200]; int temp_idx;
7490    sprintf(fn,"../t11/%s_basisfunctions_to_explore.dat",fn_prefix);
7491    f = fopen(fn, "rb");
7492    for (temp_idx=0;temp_idx<n_pairs;temp_idx++) {
```

```
7493      fread(&(basisfunctions_to_explore[temp_idx]),sizeof(struct
basisfunction),1,f);
7494    }
7495    fclose( f);
7496  }
7497
7498
7499  void write_basisfunctions_to_explore_to_file(char* fn_prefix) {
7500    FILE* f; char fn[200]; int temp_idx;
7501    sprintf(fn,"%s_basisfunctions_to_explore.dat",fn_prefix);
7502    f = fopen(fn, "wb");
7503    for (temp_idx=0;temp_idx<n_pairs;temp_idx++) {
7504      fwrite(&(basisfunctions_to_explore[temp_idx]),sizeof(struct
basisfunction),1,f);
7505    }
7506    fclose( f);
7507  }
7508
7509  void evaluate_basisfunctions_to_explore_sequential() {
7510    int j1; int j2; int v1; int v2; long long temp_idx; int count;
7511    int i;
7512    for (j1=0;j1<inputsizeinnumberofbits-1;j1++) {
7513      for (j2=j1+1;j2<inputsizeinnumberofbits;j2++) {
7514        for (v1=0;v1<=1;v1++) {
7515          for (v2=0;v2<=1;v2++) {
7516            temp_idx =
get_index_to_basisfunctions_to_explore_from_row_and_col(j1,j2);
7517            temp_idx = temp_idx*4+v1*2+v2;
7518            count = 0;
7519            for (i=0;i<NEXAMPLES_FOR_FINDING_BASISFUNCTIONS;i++) {
7520              if
(evaluate_basis_on_programinput(&(basisfunctions_to_explore[temp_idx]),manyru
ns_for_finding_basis_functions[i].inputtoprogram)) {
7521                count = count + 1;
7522              }
7523            }
7524            basisfunctions_to_explore[temp_idx].score = count;
7525          }
7526        }
7527      }
7528    }
7529  }
7530
7531  /*
7532  int*
array_with_indices_for_sorting_basisfunctions_to_explore_descending_score;
7533  int
cmpfunc_sort_array_with_indices_for_sorting_basisfunctions_to_explore_descend
ing_score(const void * a, const void * b) {
7534    int* p_a; int* p_b; int score_a; int score_b;
7535    p_a = (int*) a;
7536    p_b = (int*) b;
7537    score_a =
basisfunctions_to_explore[array_with_indices_for_sorting_basisfunctions_to_ex
plore_descending_score[*p_a]].score;
```

```
7538   score_b =
basisfunctions_to_explore[array_with_indices_for_sorting_basisfunctions_to_ex
plore_descending_score[*p_b]].score;
7539   if (score_a>score_b) {
7540     return -1;
7541   } else {
7542     if (score_a==score_b) {
7543       return 0;
7544     } else {
7545       return 1;
7546     }
7547   }
7548 }
7549
7550 void fill_basisfunctions_from_basisfunctions_to_explore() {
7551   int iterator; int it2;
7552
array_with_indices_for_sorting_basisfunctions_to_explore_descending_score =
malloc(n_pairs*sizeof(int)); if
(array_with_indices_for_sorting_basisfunctions_to_explore_descending_score==N
ULL) { printf("Error in
fill_basisfunctions_from_basisfunctions_to_explore\n"); exit(-1); }
7553   for (iterator=0;iterator<n_pairs;iterator++) {
7554
array_with_indices_for_sorting_basisfunctions_to_explore_descending_score[ite
rator] = iterator;
7555   }
7556
qsort(array_with_indices_for_sorting_basisfunctions_to_explore_descending_sco
re,
7557
n_pairs,sizeof(int),cmpfunc_sort_array_with_indices_for_sorting_basisfunction
s_to_explore_descending_score);
7558   for (iterator=0;iterator<NBASISFUNCTIONS;iterator++) {
7559     it2 =
array_with_indices_for_sorting_basisfunctions_to_explore_descending_score[ite
rator];
7560     basisfunctions[iterator] = basisfunctions_to_explore[it2];
7561     basisfunctions[iterator].coefficient = 0.0;
7562   }
7563
free(array_with_indices_for_sorting_basisfunctions_to_explore_descending_scor
e);
7564 }
7565 */
7566
7567 /*
7568 void add_element_to_index_array_end(long long* index_array,long long
index_to_add,int* p_nelements) {
7569   if ((*p_nelements)<NBASISFUNCTIONS) {
7570     index_array[(*p_nelements)] = index_to_add;
7571     (*p_nelements) = (*p_nelements) + 1;
7572   }
7573 }
7574
7575 void add_element_to_index_array_first_element(long long*
index_array,long long index_to_add,int* p_nelements) {
```

```
7576   index_array[0] = index_to_add;
7577   (*p_nelements) = 1;
7578 }
7579
7580 void add_element_to_index_array(long long* index_array,long long
index_to_add,int* p_nelements) {
7581   int position_to_insert; int temp_index;
7582   if (*p_nelements>0) {
7583     if
(basisfunctions_to_explore[index_to_add].score>basisfunctions_to_explore[inde
x_array[(*p_nelements)-1]].score) {
7584       position_to_insert = 0;
7585       while
(basisfunctions_to_explore[index_to_add].score<=basisfunctions_to_explore[ind
ex_array[position_to_insert]].score) {
7586         position_to_insert = position_to_insert + 1;
7587       }
7588       if ((*p_nelements)<NBASISFUNCTIONS) {
7589         for (temp_index=(*p_nelements)-
1;temp_index>=position_to_insert;temp_index--) {
7590           index_array[temp_index+1] = index_array[temp_index];
7591         }
7592         index_array[position_to_insert] = index_to_add;
7593         (*p_nelements) = (*p_nelements) + 1;
7594       } else {
7595         for (temp_index=NBASISFUNCTIONS-
2;temp_index>=position_to_insert;temp_index--) {
7596           index_array[temp_index+1] = index_array[temp_index];
7597         }
7598         index_array[position_to_insert] = index_to_add;
7599       }
7600     } else {
7601
add_element_to_index_array_end(index_array,index_to_add,p_nelements);
7602     }
7603   } else {
7604
add_element_to_index_array_first_element(index_array,index_to_add,p_nelements
);
7605   }
7606 }
7607
7608 void fill_basisfunctions_from_basisfunctions_to_explore() {
7609   long long iterator; int nelements; long long* index_array;
7610   index_array = malloc(NBASISFUNCTIONS*sizeof(long long)); if
(index_array==NULL) { printf("Error in
fill_basisfunctions_from_basisfunctions_to_explore\n"); exit(-1); }
7611   nelements = 0;
7612   for (iterator=0;iterator<n_pairs;iterator++) {
7613     add_element_to_index_array(index_array,iterator,&nelements); // if
it is full, then the element will not be added
7614   }
7615   for (iterator=0;iterator<NBASISFUNCTIONS;iterator++) {
7616     basisfunctions[iterator] =
basisfunctions_to_explore[index_array[iterator]];
7617     basisfunctions[iterator].coefficient = 0.0;
7618   }
```

```
7619   free(index_array);
7620 }
7621 */
7622
7623 long long*
array_with_indices_for_sorting_basisfunctions_to_explore_descending_score;
7624 int
cmpfunc_sort_array_with_indices_for_sorting_basisfunctions_to_explore_descend
ing_score(const void * a, const void * b) {
7625   long long* p_a; long long* p_b; double score_a; double score_b;
7626   p_a = (long long*) a;
7627   p_b = (long long*) b;
7628   score_a = basisfunctions_to_explore[(*p_a)].score;
7629   score_b = basisfunctions_to_explore[(*p_b)].score;
7630   if (score_a>score_b) {
7631     return -1;
7632   } else {
7633     if (score_a==score_b) {
7634       return 0;
7635     } else {
7636       return 1;
7637     }
7638   }
7639 }
7640 void fill_basisfunctions_from_basisfunctions_to_explore() {
7641   long long iterator;
7642
array_with_indices_for_sorting_basisfunctions_to_explore_descending_score =
malloc(n_pairs*sizeof(long long)); if
(array_with_indices_for_sorting_basisfunctions_to_explore_descending_score==N
ULL) { printf("Error in
fill_basisfunctions_from_basisfunctions_to_explore\n"); exit(-1); }
7643   for (iterator=0;iterator<n_pairs;iterator++) {
7644
array_with_indices_for_sorting_basisfunctions_to_explore_descending_score[ite
rator] = iterator;
7645   }
7646
qsort(array_with_indices_for_sorting_basisfunctions_to_explore_descending_sco
re,n_pairs,sizeof(long
long),cmpfunc_sort_array_with_indices_for_sorting_basisfunctions_to_explore_d
escending_score);
7647   for (iterator=0;iterator<NBASISFUNCTIONS;iterator++) {
7648     basisfunctions[iterator] =
basisfunctions_to_explore[array_with_indices_for_sorting_basisfunctions_to_ex
plore_descending_score[iterator]];
7649     basisfunctions[iterator].coefficient = 0.0;
7650   }
7651
free(array_with_indices_for_sorting_basisfunctions_to_explore_descending_scor
e);
7652 }
7653
7654 void
generate_basis_functions_without_coefficients_select_among_all_pairs(char*
fn_prefix,int n_examples_in_file) {
```

```
7655   int status_clock_gettime_begin; int status_clock_gettime_end; struct
timespec mybegin; struct timespec myend;
7656
allocate_memory_for_runs_used_for_finding_basis_functions(NEXAMPLES_FOR_FINDI
NG_BASISFUNCTIONS);
7657
fill_memory_for_runs_used_for_finding_basis_functions(NEXAMPLES_FOR_FINDING_B
ASISFUNCTIONS,n_examples_in_file);
7658
7659   n_pairs = (inputsizeinnumberofbits*(inputsizeinnumberofbits-1)/2)*4;
7660   basisfunctions_to_explore = malloc(n_pairs*sizeof(struct
basisfunction));
7661   if (basisfunctions_to_explore==NULL) { printf("Error after malloc.
basisfunctions_to_explore\n"); exit(-1); }
7662     status_clock_gettime_begin = clock_gettime(CLOCK_MONOTONIC_RAW,
&mybegin); if (status_clock_gettime_begin!=0) { printf("Error in
run_program_with_input_dont_use_filesystem.
status_clock_gettime_begin=%d\n",status_clock_gettime_begin); exit(-1); }
7663   fill_basisfunctions_to_explore_sequential();
7664     status_clock_gettime_end = clock_gettime(CLOCK_MONOTONIC_RAW,
&myend); if (status_clock_gettime_end!=0) { printf("Error in
run_program_with_input_dont_use_filesystem.
status_clock_gettime_end=%d\n",status_clock_gettime_end); exit(-1); }
7665
write_single_double_to_file(fn_prefix,"time_for_fill_basisfunctions_to_explor
e_sequential.txt",diff_s_timespec(&myend, &mybegin));
7666     status_clock_gettime_begin = clock_gettime(CLOCK_MONOTONIC_RAW,
&mybegin); if (status_clock_gettime_begin!=0) { printf("Error in
run_program_with_input_dont_use_filesystem.
status_clock_gettime_begin=%d\n",status_clock_gettime_begin); exit(-1); }
7667   evaluate_basisfunctions_to_explore_parallel();
7668   write_basisfunctions_to_explore_to_file(fn_prefix);
7669   read_basisfunctions_to_explore_from_file(fn_prefix);
7670     status_clock_gettime_end = clock_gettime(CLOCK_MONOTONIC_RAW,
&myend); if (status_clock_gettime_end!=0) { printf("Error in
run_program_with_input_dont_use_filesystem.
status_clock_gettime_end=%d\n",status_clock_gettime_end); exit(-1); }
7671
write_single_double_to_file(fn_prefix,"time_for_evaluate_basisfunctions_to_ex
plore_parallel.txt",diff_s_timespec(&myend, &mybegin));
7672     status_clock_gettime_begin = clock_gettime(CLOCK_MONOTONIC_RAW,
&mybegin); if (status_clock_gettime_begin!=0) { printf("Error in
run_program_with_input_dont_use_filesystem.
status_clock_gettime_begin=%d\n",status_clock_gettime_begin); exit(-1); }
7673   fill_basisfunctions_from_basisfunctions_to_explore();
7674     status_clock_gettime_end = clock_gettime(CLOCK_MONOTONIC_RAW,
&myend); if (status_clock_gettime_end!=0) { printf("Error in
run_program_with_input_dont_use_filesystem.
status_clock_gettime_end=%d\n",status_clock_gettime_end); exit(-1); }
7675
write_single_double_to_file(fn_prefix,"time_for_fill_basisfunctions_from_basi
sfunctions_to_explore.txt",diff_s_timespec(&myend, &mybegin));
7676
7677   free(basisfunctions_to_explore);
7678
free_memory_for_runs_used_for_finding_basis_functions(NEXAMPLES_FOR_FINDING_B
ASISFUNCTIONS);
```

```
7679 }
7680
7681 void
generate_basis_functions_without_coefficients_select_among_all_pairs_temp(cha
r* fn_prefix,int n_examples_in_file) {
7682   int status_clock_gettime_begin; int status_clock_gettime_end; struct
timespec mybegin; struct timespec myend;
7683   //
allocate_memory_for_runs_used_for_finding_basis_functions(NEXAMPLES_FOR_FINDI
NG_BASISFUNCTIONS);
7684   //
fill_memory_for_runs_used_for_finding_basis_functions(NEXAMPLES_FOR_FINDING_B
ASISFUNCTIONS,n_examples_in_file);
7685
7686   n_pairs = (inputsizeinnumberofbits*(inputsizeinnumberofbits-1)/2)*4;
7687   basisfunctions_to_explore = malloc(n_pairs*sizeof(struct
basisfunction));
7688   // if (basisfunctions_to_explore==NULL) { printf("Error after malloc.
basisfunctions_to_explore\n"); exit(-1); }
7689   //  status_clock_gettime_begin = clock_gettime(CLOCK_MONOTONIC_RAW,
&mybegin); if (status_clock_gettime_begin!=0) { printf("Error in
run_program_with_input_dont_use_filesystem.
status_clock_gettime_begin=%d\n",status_clock_gettime_begin); exit(-1); }
7690   // fill_basisfunctions_to_explore_sequential();
7691   //  status_clock_gettime_end = clock_gettime(CLOCK_MONOTONIC_RAW,
&myend); if (status_clock_gettime_end!=0) { printf("Error in
run_program_with_input_dont_use_filesystem.
status_clock_gettime_end=%d\n",status_clock_gettime_end); exit(-1); }
7692   //
write_single_double_to_file(fn_prefix,"time_for_fill_basisfunctions_to_explor
e_sequential.txt",diff_s_timespec(&myend, &mybegin));
7693   //  status_clock_gettime_begin = clock_gettime(CLOCK_MONOTONIC_RAW,
&mybegin); if (status_clock_gettime_begin!=0) { printf("Error in
run_program_with_input_dont_use_filesystem.
status_clock_gettime_begin=%d\n",status_clock_gettime_begin); exit(-1); }
7694   // evaluate_basisfunctions_to_explore_parallel();
7695   // write_basisfunctions_to_explore_to_file(fn_prefix);
7696   // read_basisfunctions_to_explore_from_file(fn_prefix);
7697   read_basisfunctions_to_explore_from_file_temp(fn_prefix);
7698   //  status_clock_gettime_end = clock_gettime(CLOCK_MONOTONIC_RAW,
&myend); if (status_clock_gettime_end!=0) { printf("Error in
run_program_with_input_dont_use_filesystem.
status_clock_gettime_end=%d\n",status_clock_gettime_end); exit(-1); }
7699   //
write_single_double_to_file(fn_prefix,"time_for_evaluate_basisfunctions_to_ex
plore_parallel.txt",diff_s_timespec(&myend, &mybegin));
7700   //  status_clock_gettime_begin = clock_gettime(CLOCK_MONOTONIC_RAW,
&mybegin); if (status_clock_gettime_begin!=0) { printf("Error in
run_program_with_input_dont_use_filesystem.
status_clock_gettime_begin=%d\n",status_clock_gettime_begin); exit(-1); }
7701   fill_basisfunctions_from_basisfunctions_to_explore();
7702   //  status_clock_gettime_end = clock_gettime(CLOCK_MONOTONIC_RAW,
&myend); if (status_clock_gettime_end!=0) { printf("Error in
run_program_with_input_dont_use_filesystem.
status_clock_gettime_end=%d\n",status_clock_gettime_end); exit(-1); }
```

```
7703   //
write_single_double_to_file(fn_prefix,"time_for_fill_basisfunctions_from_basi
sfunctions_to_explore.txt",diff_s_timespec(&myend, &mybegin));
7704
7705   free(basisfunctions_to_explore);
7706   //
free_memory_for_runs_used_for_finding_basis_functions(NEXAMPLES_FOR_FINDING_B
ASISFUNCTIONS);
7707 }
7708
7709 double compute_max_abs_error(int n_examples_in_file) {
7710   int i; double max_abs_error;
7711   max_abs_error = fabs(manyruns[0].error);
7712   for (i=1;i<n_examples_in_file;i++) {
7713     if (max_abs_error<fabs(manyruns[i].error)) {
7714       max_abs_error = fabs(manyruns[i].error);
7715     }
7716   }
7717   return max_abs_error;
7718 }
7719
7720 void print_derivative_of_loss_wrt_k_to_file(int iterator) {
7721   int k; FILE* f; char fn[200];
7722   sprintf(fn,"intermediate_%d_derivative_of_loss_wrt_k",iterator);
7723   f = fopen(fn, "w+"); if (f==NULL) { printf("Error in
print_derivative_of_loss_wrt_k_to_file\n"); exit(-1); }
7724   for (k=0;k<NBASISFUNCTIONS;k++) {
7725     fprintf(f,"%21.15lf\n",derivative_of_loss_wrt_k[k]);
7726   }
7727   fclose( f);
7728 }
7729
7730 void print_basisfunctions_coefficient(int iterator) {
7731   int k; FILE* f; char fn[200];
7732   sprintf(fn,"intermediate_%d_basisfunctions_coefficient",iterator);
7733   f = fopen(fn, "w+"); if (f==NULL) { printf("Error in
print_basisfunctions_coefficient\n"); exit(-1); }
7734   for (k=0;k<NBASISFUNCTIONS;k++) {
7735     fprintf(f,"%21.15lf\n",basisfunctions[k].coefficient);
7736   }
7737   fclose( f);
7738 }
7739
7740 void print_weights_for_buckets(int iterator) {
7741   int b; FILE* f; char fn[200];
7742   sprintf(fn,"intermediate_%d_weights_for_buckets",iterator);
7743   f = fopen(fn, "w+"); if (f==NULL) { printf("Error in
print_weights_for_buckets\n"); exit(-1); }
7744   for (b=0;b<NBUCKETS_USED_FOR_WEIGHING;b++) {
7745     fprintf(f,"%21.15lf\n",buckets_weights[b]);
7746   }
7747   fclose( f);
7748 }
7749
7750 void find_WCET_v4(char* fn_with_executiontimemeasurements_inputs,char*
fn_with_executiontimemeasurements_times,char* fn_prefix,int programid,int
n_examples_in_file) {
```

```
7751    char fn[2000]; char fn2[2000]; double t0; double t1; FILE* f;
7752    char temp1_fn_prefix[200];
7753    char temp2_fn_prefix[200];
7754    int iterator;
7755    int status_clock_gettime_begin; int status_clock_gettime_end; struct
timespec mybegin; struct timespec myend;
7756    double best_alpha; double best_max_abs_error; double
current_max_abs_error; double alpha;
7757
allocate_memory_for_predictedworstcaseinput_phase0_and_predictedworstcaseinpu
t_phase1();
7758    allocate_memory_for_learnable_weights_v2(); // this allocates memory
for w0 and b56 which we need; we don't need the others that are allocated by
this func
7759    allocate_memory_for_manyruns_v2(n_examples_in_file);
7760
read_captured_data_from_disk_two_explicit_fn(fn_with_executiontimemeasurement
s_inputs,fn_with_executiontimemeasurements_times,n_examples_in_file);
7761
calculate_min_max_average_executiontime_from_manyruns(n_examples_in_file);
7762    printf("n_examples_in_file = %d\n",n_examples_in_file);
fflush(stdout);
7763    printf("min_observed_time = %lf\n",min_observed_time);
fflush(stdout);
7764    printf("max_observed_time = %lf\n",max_observed_time);
fflush(stdout);
7765    sprintf(temp1_fn_prefix,"%s_for_affine_model_1",fn_prefix);
7766    read_w0_b56_from_textfile(temp1_fn_prefix);
7767
7768    // compute_error_and_prediction_based_on_w0_b56(n_examples_in_file);
7769    // (*p_b56_value) = (*p_b56_value) -
compute_average_error_without_weight(n_examples_in_file);
7770
7771    //
train_affine_model_initialize_w0_and_b56_use_weighting_with_buckets(n_example
s_in_file);
7772    // for (iterator=0;iterator<80;iterator++) {
7773    //
train_affine_model_adjust_w0_use_weighting_with_buckets(n_examples_in_file);
7774    // }
7775
7776    sprintf(temp1_fn_prefix,"%s_for_affine_model_1",fn_prefix);
7777    compute_error_and_prediction_based_on_w0_b56(n_examples_in_file);
7778    write_all_predictions_to_file(temp1_fn_prefix,n_examples_in_file);
7779    write_all_errors_to_file(temp1_fn_prefix,n_examples_in_file);
7780
write_summary_of_predictions_and_errors_to_files(temp1_fn_prefix,n_examples_i
n_file);
7781
7782    write_w0_b56_to_textfile(temp1_fn_prefix);
7783    read_w0_b56_from_textfile(temp1_fn_prefix);
7784
find_WCET_input_from_prediction_model_w0_b56(temp1_fn_prefix,predictedworstca
seinput,&predicted_time_predictedworstcaseinput);
7785
copy_inputtoprogram(predictedworstcaseinput_phase0,predictedworstcaseinput);
```

```
7786   predicted_time_predictedworstcaseinput_phase0 =
predicted_time_predictedworstcaseinput;
7787
7788   // generate_basis_functions_without_coefficients();
7789   //
generate_basis_functions_without_coefficients_select_among_all_pairs(fn_prefi
x,n_examples_in_file);
7790
sprintf(temp2_fn_prefix,"%s_for_affine_model_with_basis_functions_2",fn_prefi
x);
7791   read_basis_functions_from_file(temp2_fn_prefix);
7792   set_all_coefficients_of_basis_functions_to_zero();
7793
7794   // train_model_with_basis_functions_adjust_w5(n_examples_in_file);
7795   // train_model_with_basis_functions_adjust_w5(n_examples_in_file);
7796
7797   status_clock_gettime_begin = clock_gettime(CLOCK_MONOTONIC_RAW,
&mybegin); if (status_clock_gettime_begin!=0) { printf("Error in
run_program_with_input_dont_use_filesystem.
status_clock_gettime_begin=%d\n",status_clock_gettime_begin); exit(-1); }
7798   // for (iterator=0;iterator<80;iterator++) {
7799   // for (iterator=0;iterator<1;iterator++) {
7800   // for (iterator=0;iterator<10;iterator++) {
7801   // for (iterator=0;iterator<1;iterator++) {
7802   // for (iterator=0;iterator<10;iterator++) {
7803   // for (iterator=0;iterator<80;iterator++) {
7804   // for (iterator=0;iterator<0;iterator++) {
7805   // for (iterator=0;iterator<1;iterator++) {
7806   // for (iterator=0;iterator<2;iterator++) {
7807   // for (iterator=0;iterator<3;iterator++) {
7808   // for (iterator=0;iterator<4;iterator++) {
7809   // for (iterator=0;iterator<5;iterator++) {
7810   // for (iterator=0;iterator<10;iterator++) {
7811   // for (iterator=0;iterator<100;iterator++) {
7812   // for (iterator=0;iterator<300;iterator++) {
7813   // for (iterator=0;iterator<1;iterator++) {
7814   // for (iterator=0;iterator<100;iterator++) {
7815   // for (iterator=0;iterator<1;iterator++) {
7816   // for (iterator=0;iterator<3;iterator++) {
7817   // for (iterator=0;iterator<1;iterator++) {
7818   // for (iterator=0;iterator<2;iterator++) {
7819   // for (iterator=0;iterator<20;iterator++) {
7820   // for (iterator=0;iterator<100;iterator++) {
7821   // for (iterator=0;iterator<1;iterator++) {
7822   // for (iterator=0;iterator<100;iterator++) {
7823   // for (iterator=0;iterator<1;iterator++) {
7824   // for (iterator=0;iterator<30;iterator++) {
7825   // for (iterator=0;iterator<1000;iterator++) {
7826   // for (iterator=0;iterator<60;iterator++) {
7827   // for (iterator=0;iterator<2000;iterator++) {
7828   // for (iterator=0;iterator<1;iterator++) {
7829   // for (iterator=0;iterator<4;iterator++) {
7830   // for (iterator=0;iterator<1;iterator++) {
7831   // for (iterator=0;iterator<3;iterator++) {
7832   for (iterator=0;iterator<10;iterator++) {
7833     save_all_coefficients_of_basis_functions();
7834     best_max_abs_error = -1.0;
```

```
7835      best_alpha = -1.0;
7836      current_max_abs_error = -1.0;
7837      // alpha = 1.000;
7838      alpha = 0.1;
7839      // while (alpha>0.000000001) {
7840      // while (alpha>0.000000000001) {
7841      // while (alpha>0.000001) {
7842      while (alpha>0.000000001) {
7843
train_model_with_basis_functions_adjust_w5_use_weighting_with_buckets_horizon
tal_scan(n_examples_in_file,alpha);
7844
compute_error_and_prediction_based_on_w0_b56_and_basis_functions(n_examples_i
n_file);
7845        current_max_abs_error = compute_max_abs_error(n_examples_in_file);
7846        if (best_alpha==-1.0) {
7847          best_max_abs_error = current_max_abs_error;
7848          best_alpha = alpha;
7849        } else {
7850           if (best_max_abs_error>current_max_abs_error) {
7851              best_max_abs_error = current_max_abs_error;
7852              best_alpha = alpha;
7853           }
7854        }
7855        alpha = alpha / 2.0;
7856        restore_all_coefficients_of_basis_functions();
7857      }
7858      printf("iteration=%d best_max_abs_error=%25.21lf
best_alpha=%25.21lf\n",iterator,best_max_abs_error,best_alpha);
7859      // restore_all_coefficients_of_basis_functions();
7860      alpha = best_alpha;
7861
train_model_with_basis_functions_adjust_w5_use_weighting_with_buckets_horizon
tal_scan(n_examples_in_file,alpha);
7862      print_derivative_of_loss_wrt_k_to_file(iterator);
7863      print_basisfunctions_coefficient(iterator);
7864      print_weights_for_buckets(iterator);
7865
compute_error_and_prediction_based_on_w0_b56_and_basis_functions(n_examples_i
n_file);
7866    }
7867    // we can do 17 calls to
train_model_with_basis_functions_adjust_w5_use_weighting_with_buckets_horizon
tal_scan and compute_error_and_prediction_based_on_w0_b56_and_basis_functions
per hour
7868    // hence, in the next 5 days, we can do 2000 calls. Since we have 30
different alphas, we should run the outer for loops for 2000/30 iterations;
that is 60 iterations.
7869    // since we only train the coefficients for the basis functions, we
can probably do many more; e.g., 32768/1000=32 times more. Then we end up
with 2000 iterations for the outer loop.
7870    // this should take approximately 15*1.5 hours; that is 20 hours
7871    // this run with iterator<10 should take approximately 25 hours
7872
7873    status_clock_gettime_end = clock_gettime(CLOCK_MONOTONIC_RAW, &myend);
if (status_clock_gettime_end!=0) { printf("Error in
```

```
run_program_with_input_dont_use_filesystem.
status_clock_gettime_end=%d\n",status_clock_gettime_end); exit(-1); }
7874   //
write_single_double_to_file(fn_prefix,"time_for_train_affine_model_adjust_w5_
use_weighting_with_buckets.txt",diff_s_timespec(&myend, &mybegin));
7875
write_single_double_to_file(fn_prefix,"time_for_train_model_with_basis_functi
ons_adjust_w5_use_weighting_with_buckets.txt",diff_s_timespec(&myend,
&mybegin));
7876
7877
sprintf(temp2_fn_prefix,"%s_for_affine_model_with_basis_functions_2",fn_prefi
x);
7878
compute_error_and_prediction_based_on_w0_b56_and_basis_functions(n_examples_i
n_file);
7879   write_all_predictions_to_file(temp2_fn_prefix,n_examples_in_file);
7880   write_all_errors_to_file(temp2_fn_prefix,n_examples_in_file);
7881
write_summary_of_predictions_and_errors_to_files(temp2_fn_prefix,n_examples_i
n_file);
7882
7883   print_basis_functions_to_file(temp2_fn_prefix);
7884   read_basis_functions_from_file(temp2_fn_prefix);
7885
find_WCET_input_from_prediction_model_w0_b56_w5(temp2_fn_prefix,predictedwors
tcaseinput,&predicted_time_predictedworstcaseinput);
7886
copy_inputtoprogram(predictedworstcaseinput_phase1,predictedworstcaseinput);
7887   predicted_time_predictedworstcaseinput_phase1 =
predicted_time_predictedworstcaseinput;
7888
7889   printf("Running with obtained worst-case input\n");
7890   sleep(10);
7891   setprocessoraffinity_to_allow_just_a_single_processor_proc0();
7892   sleep(60);
7893
run_program_with_two_different_inputs_dont_use_filesystem_run_X_times(predict
edworstcaseinput_phase0,predictedworstcaseinput_phase1,programid,100,&t0,&t1)
;
7894
7895   sprintf(fn, "%s_0_predictedworstcaseinput.dat",        fn_prefix);
7896   sprintf(fn2,"%s_0_predictedworstcaseexecutiontime.dat",fn_prefix);
7897
write_input_and_execution_time_to_files(fn,fn2,predictedworstcaseinput_phase0
,t0);
7898
sprintf(fn,"%s_0_predictedworstcaseexecutiontime_notrun.dat",fn_prefix); f =
fopen(fn, "w+"); fprintf( f, "%18.12lf\n",
predicted_time_predictedworstcaseinput_phase0); fclose( f);
7899
7900   sprintf(fn, "%s_1_predictedworstcaseinput.dat",        fn_prefix);
7901   sprintf(fn2,"%s_1_predictedworstcaseexecutiontime.dat",fn_prefix);
7902
write_input_and_execution_time_to_files(fn,fn2,predictedworstcaseinput_phase1
,t1);
```

```
7903
sprintf(fn,"%s_1_predictedworstcaseexecutiontime_notrun.dat",fn_prefix); f =
fopen(fn, "w+"); fprintf( f, "%18.12lf\n",
predicted_time_predictedworstcaseinput_phase1); fclose( f);
7904
7905   setprocessoraffinity_to_allow_all_processors(); // we don't need to
restore this but it does not hurt
7906
7907   free_memory_for_manyruns_v2(n_examples_in_file);
7908   free_memory_for_learnable_weights_v2(); // we only need w0 and b56
7909
free_memory_for_predictedworstcaseinput_phase0_and_predictedworstcaseinput_ph
ase1();
7910 }
7911
7912 void
fill_derivative_of_loss_wrt_k_considering_minimize_max_abs_error(int
index_with_max_abs_error) {
7913   int k;
7914   for (k=0;k<NBASISFUNCTIONS;k++) {
7915     if (manyruns[index_with_max_abs_error].error>=0) {
7916       derivative_of_loss_wrt_k[k] =
evaluate_basis_on_programinput(&(basisfunctions[k]),manyruns[index_with_max_a
bs_error].inputtoprogram);
7917     } else {
7918       derivative_of_loss_wrt_k[k] = -
evaluate_basis_on_programinput(&(basisfunctions[k]),manyruns[index_with_max_a
bs_error].inputtoprogram);
7919     }
7920   }
7921 }
7922
7923 // double compute_max_abs_w0() {
7924 //   int j; double max_so_far;
7925 //   max_so_far = fabs(p_w0_value[0]);
7926 //   for (j=1;j<inputsizeinnumberofbits;j++) {
7927 //     if (max_so_far<fabs(p_w0_value[j])) {
7928 //       max_so_far = fabs(p_w0_value[j]);
7929 //     }
7930 //   }
7931 //   return max_so_far;
7932 // }
7933
7934 void change_all_coefficients_of_basis_functions(double alpha) {
7935   int k;
7936   for (k=0;k<NBASISFUNCTIONS;k++) {
7937     basisfunctions[k].coefficient = basisfunctions[k].coefficient -
alpha * derivative_of_loss_wrt_k[k];
7938   }
7939 }
7940
7941 void compute_index_with_max_abs_error_and_max_abs_error(int
n_examples,int* p_index_with_max_abs_error,double* p_max_abs_error) {
7942   int row_in_dataset;
7943   row_in_dataset=0;
7944   *p_index_with_max_abs_error = row_in_dataset;
7945   *p_max_abs_error            = fabs(manyruns[row_in_dataset].error);
```

```
  for (row_in_dataset=1;row_in_dataset<n_examples;row_in_dataset++) {
    if ((*p_max_abs_error)<fabs(manyruns[row_in_dataset].error)) {
      *p_index_with_max_abs_error = row_in_dataset;
      *p_max_abs_error            =
fabs(manyruns[row_in_dataset].error);
    }
  }
}

/*
void divide_alpha(int iterator,double* p_alpha) {
  if        (iterator==0) {
    (*p_alpha) = (*p_alpha)/1000.0;
  } else if (iterator==1) {
    (*p_alpha) = (*p_alpha)/300.0;
  } else if (iterator==2) {
    (*p_alpha) = (*p_alpha)/100.0;
  } else if (iterator==3) {
    (*p_alpha) = (*p_alpha)/30.0;
  } else if (iterator==4) {
    (*p_alpha) = (*p_alpha)/10.0;
  } else {
    (*p_alpha) = (*p_alpha)/3.0;
  }
}
void multiply_alpha(int iterator,double* p_alpha) {
  if        (iterator==0) {
    (*p_alpha) = (*p_alpha)*300.0;
  } else if (iterator==1) {
    (*p_alpha) = (*p_alpha)*100.0;
  } else if (iterator==2) {
    (*p_alpha) = (*p_alpha)*30.0;
  } else if (iterator==3) {
    (*p_alpha) = (*p_alpha)*10.0;
  } else if (iterator==4) {
    (*p_alpha) = (*p_alpha)*3.0;
  } else {
    (*p_alpha) = (*p_alpha)*1.1;
  }
}
*/

void divide_alpha(int iterator,double* p_alpha) {
  if        (iterator==0) {
    (*p_alpha) = (*p_alpha)/10.0;
  } else if (iterator==1) {
    (*p_alpha) = (*p_alpha)/7.0;
  } else if (iterator==2) {
    (*p_alpha) = (*p_alpha)/5.0;
  } else if (iterator==3) {
    (*p_alpha) = (*p_alpha)/3.0;
  } else if (iterator==4) {
    (*p_alpha) = (*p_alpha)/2.0;
  } else {
    (*p_alpha) = (*p_alpha)/1.6;
  }
}
```

```
8002 void multiply_alpha(int iterator,double* p_alpha) {
8003   if        (iterator==0) {
8004     (*p_alpha) = (*p_alpha)*7.0;
8005   } else if (iterator==1) {
8006     (*p_alpha) = (*p_alpha)*5.0;
8007   } else if (iterator==2) {
8008     (*p_alpha) = (*p_alpha)*3.0;
8009   } else if (iterator==3) {
8010     (*p_alpha) = (*p_alpha)*2.0;
8011   } else if (iterator==4) {
8012     (*p_alpha) = (*p_alpha)*1.6;
8013   } else {
8014     (*p_alpha) = (*p_alpha)*1.1;
8015   }
8016 }
8017
8018 // struct table_element {
8019 //   int sample_size;
8020 //   double alpha;
8021 //   double best_alpha;
8022 //   double loss;
8023 //   double loss_best_alpha;
8024 // };
8025
8026 // int n_elements_table_with_different_sample_size;
8027 // struct table_element table_with_different_sample_size[100;]
8028
8029 // void fill_table_with_different_sample_size(int n_examples_in_file) {
8030 //   int iterator; int temp;
8031 //   n_elements_table_with_different_sample_size = 0;
8032 //   temp = 1;
8033 //   for (iterator=0;iterator<9;iterator++) {
8034 //     table_element table_with_different_sample_size[iterator] = temp;
8035 //     temp = temp * 8;
8036 //     if (sample_size>n_examples_in_file) { sample_size =
n_examples_in_file; }
8037 //     table_element table_with_different_sample_size = table_element
table_with_different_sample_size + 1;
8038 //   }
8039 // }
8040
8041 /*
8042 // it is assumed that an affine model has already been obtained, all
coefficients for basis functions are already zero.
8043 // it is also assumed that the following has been computed: pred and
error for all examples.
8044 void
train_model_with_basis_functions_adjust_w5_minimize_max_abs_error(char*
fn_prefix,int n_examples_in_file) {
8045   int index_with_max_abs_error; double max_abs_error; double alpha;
double old_max_abs_error; int iterator;
8046
compute_index_with_max_abs_error_and_max_abs_error(n_examples_in_file,&index_
with_max_abs_error,&max_abs_error);
8047
fill_derivative_of_loss_wrt_k_considering_minimize_max_abs_error(index_with_m
ax_abs_error);
```

```
8048   alpha = 1.0;
8049   // for (iterator=0;iterator<10;iterator++) {
8050   // for (iterator=0;iterator<100;iterator++) { // this will take
approximately 15 hours
8051   // for (iterator=0;iterator<20;iterator++) {
8052   // for (iterator=0;iterator<1;iterator++) { // this will take
approximately 30 hours
8053   // for (iterator=0;iterator<20;iterator++) {  // this will take
approximately 7 hours
8054   // for (iterator=0;(iterator<60) &&
(alpha>0.000000000000001);iterator++) {  // this will take approximately 7
hours
8055   for (iterator=0;(iterator<5) && (alpha>0.000000000000001);iterator++)
{  // this will take approximately 7 hours
8056     printf("iterator = %d\n",iterator);
8057     printf("  alpha = %21.18lf max_abs_error = %21.18lf
index_with_max_abs_error=%d\n",alpha,max_abs_error,index_with_max_abs_error);
8058     old_max_abs_error = max_abs_error;
8059     save_all_coefficients_of_basis_functions();
8060     change_all_coefficients_of_basis_functions(alpha);
8061
compute_error_and_prediction_based_on_w0_b56_and_basis_functions(n_examples_i
n_file);
8062
compute_index_with_max_abs_error_and_max_abs_error(n_examples_in_file,&index_
with_max_abs_error,&max_abs_error);
8063     while ((max_abs_error>=old_max_abs_error) &&
(alpha>0.000000000000001)) {
8064       printf("    alpha = %21.18lf max_abs_error = %21.18lf
index_with_max_abs_error=%d\n",alpha,max_abs_error,index_with_max_abs_error);
8065       restore_all_coefficients_of_basis_functions();
8066       divide_alpha(iterator,&alpha);
8067       change_all_coefficients_of_basis_functions(alpha);
8068
compute_error_and_prediction_based_on_w0_b56_and_basis_functions(n_examples_i
n_file);
8069
compute_index_with_max_abs_error_and_max_abs_error(n_examples_in_file,&index_
with_max_abs_error,&max_abs_error);
8070     }
8071     printf("    alpha = %21.18lf max_abs_error = %21.18lf
index_with_max_abs_error=%d\n",alpha,max_abs_error,index_with_max_abs_error);
8072     print_derivative_of_loss_wrt_k_to_file(iterator);
8073     print_basisfunctions_coefficient(iterator);
8074
fill_derivative_of_loss_wrt_k_considering_minimize_max_abs_error(index_with_m
ax_abs_error);
8075     multiply_alpha(iterator,&alpha);
8076   }
8077 }
8078 */
8079
8080 void fill_index_of_examples_descending_abs_error(int n_examples_in_file)
{
8081   int iterator;
8082   for (iterator=0;iterator<n_examples_in_file;iterator++) {
```

```
8083      array_with_indices_for_sorting_descending_abserror[iterator] =
iterator;
8084    }
8085
qsort(array_with_indices_for_sorting_descending_abserror,n_examples_in_file,s
izeof(int),cmpfunc_sort_array_with_indices_for_sorting_descending_abserror);
8086 }
8087
8088 struct tableitem {
8089   int nsamples;
8090   double alpha;
8091   double max_abs_error;
8092   char fn_basis_coefficients[400]; // note that this is actually not the
filename; the filename that we will use will concatenate "_basis_functions"
8093 };
8094
8095 double get_span_of_nsamples_largest_abs_error(int nsamples) {
8096   int index_a; int index_b; double abs_err_a; double abs_err_b;
8097   index_a = array_with_indices_for_sorting_descending_abserror[0];
8098   index_b = array_with_indices_for_sorting_descending_abserror[nsamples-
1];
8099   abs_err_a = fabs(manyruns[index_a].error);
8100   abs_err_b = fabs(manyruns[index_b].error);
8101
8102   printf("In get_span_of_nsamples_largest_abs_error. abs_err_a =
%21.18lf abs_err_b = %21.18lf\n",abs_err_a,abs_err_b);
8103
8104   return abs_err_a - abs_err_b;
8105 }
8106
8107 int ntableitems;
8108 struct tableitem tableitems[100];
8109
8110 void initialize_tableitems(char* prefix,int n_examples_in_file) {
8111   int exceeded; int go_on; int nsamples;
8112   array_with_indices_for_sorting_descending_abserror =
malloc(n_examples_in_file*sizeof(int)); if
(array_with_indices_for_sorting_descending_abserror==NULL) { printf("Error in
initialize_tableitems\n"); exit(-1); }
8113   fill_index_of_examples_descending_abs_error(n_examples_in_file);
8114   exceeded = 0; go_on = 1; nsamples = 1;
8115   ntableitems = 0;
8116   while (go_on) {
8117     tableitems[ntableitems].nsamples = nsamples;
8118     tableitems[ntableitems].alpha =
get_span_of_nsamples_largest_abs_error(nsamples+1)/NBASISFUNCTIONS; // this
will give us approximately alpha := 1 * 10^{-6} / ( 10 * 10^{3}) = 0.1 *
10^{-9}
8119
sprintf(tableitems[ntableitems].fn_basis_coefficients,"%s_tableitems_%d_coeff
icients_%d",prefix,ntableitems,nsamples);
8120
8121     printf("In initialize_tableitems. nsamples = %d
get_span_of_nsamples_largest_abs_error(nsamples+1) =
%21.18lf\n",nsamples,get_span_of_nsamples_largest_abs_error(nsamples+1));
8122
8123     ntableitems = ntableitems + 1;
```

```
8124      nsamples = nsamples+1;
8125      if (nsamples>=10) {
8126        if (exceeded==0) {
8127          nsamples = 10;
8128          exceeded = 1;
8129        } else {
8130          nsamples = n_examples_in_file;
8131          go_on = 0;
8132        }
8133      }
8134    }
8135    free(array_with_indices_for_sorting_descending_abserror);
8136  }
8137
8138  void print_tableitems() {
8139    int i;
8140    for (i=0;i<ntableitems;i++) {
8141      printf("tableitems[%d].nsamples = %d\n",
i,tableitems[i].nsamples);
8142      printf("tableitems[%d].alpha  =   %21.18lf\n",
i,tableitems[i].alpha);
8143      printf("tableitems[%d].fn_basis_coefficients = %s\n",
i,tableitems[i].fn_basis_coefficients);
8144    }
8145  }
8146
8147
8148  void copy_alpha_from_first_tableitem_to_others() {
8149    int i;
8150    for (i=1;i<ntableitems;i++) {
8151      tableitems[i].alpha = tableitems[0].alpha;
8152    }
8153  }
8154
8155  void make_all_except_first_tableitem_very_high_max_abs_error() {
8156    int i;
8157    for (i=1;i<ntableitems;i++) {
8158      tableitems[i].max_abs_error = tableitems[0].max_abs_error * 1000;
8159    }
8160  }
8161
8162  void
fill_derivative_of_loss_wrt_k_considering_minimize_max_abs_error_from_samples
(int nsamples) {
8163    int k; int it; int index;
8164    for (k=0;k<NBASISFUNCTIONS;k++) {
8165      derivative_of_loss_wrt_k[k] = 0.0;
8166    }
8167    for (it=0;it<nsamples;it++) {
8168      index = array_with_indices_for_sorting_descending_abserror[it];
8169      for (k=0;k<NBASISFUNCTIONS;k++) {
8170        if (manyruns[index].error>=0) {
8171          derivative_of_loss_wrt_k[k] = derivative_of_loss_wrt_k[k] +
evaluate_basis_on_programinput(&(basisfunctions[k]),manyruns[index].inputtopr
ogram);
8172        } else {
```

```
8173          derivative_of_loss_wrt_k[k] = derivative_of_loss_wrt_k[k] -
evaluate_basis_on_programinput(&(basisfunctions[k]),manyruns[index].inputtopr
ogram);
8174        }
8175      }
8176    }
8177    for (k=0;k<NBASISFUNCTIONS;k++) {
8178      derivative_of_loss_wrt_k[k] = derivative_of_loss_wrt_k[k]/nsamples;
8179    }
8180  }
8181
8182  void print_derivative_of_loss_wrt_k_to_file_iterator_and_i(int
iterator,int i) {
8183    int k; FILE* f; char fn[200];
8184    sprintf(fn,"intermediate_%d_%d_derivative_of_loss_wrt_k",iterator,i);
8185    f = fopen(fn, "w+"); if (f==NULL) { printf("Error in
print_derivative_of_loss_wrt_k_to_file_iterator_and_i\n"); exit(-1); }
8186    for (k=0;k<NBASISFUNCTIONS;k++) {
8187      fprintf(f,"%21.15lf\n",derivative_of_loss_wrt_k[k]);
8188    }
8189    fclose( f);
8190  }
8191
8192
8193  void
print_derivative_of_loss_wrt_k_to_file_iterator_and_nsamples_and_iterator2(in
t iterator,int nsamples,int iterator2) {
8194    int k; FILE* f; char fn[200];
8195
sprintf(fn,"intermediate_%d_%d_%d_derivative_of_loss_wrt_k",iterator,nsamples
,iterator2);
8196    f = fopen(fn, "w+"); if (f==NULL) { printf("Error in
print_derivative_of_loss_wrt_k_to_file_iterator_and_i\n"); exit(-1); }
8197    for (k=0;k<NBASISFUNCTIONS;k++) {
8198      fprintf(f,"%21.15lf\n",derivative_of_loss_wrt_k[k]);
8199    }
8200    fclose( f);
8201  }
8202
8203  void fill_tableitems(int n_examples_in_file,int iterator) {
8204    int i; double max_abs_error; double old_max_abs_error; double alpha;
int index_with_max_abs_error;
8205    int to_use_ntableitems;
8206    array_with_indices_for_sorting_descending_abserror =
malloc(n_examples_in_file*sizeof(int)); if
(array_with_indices_for_sorting_descending_abserror==NULL) { printf("Error in
fill_tableitems\n"); exit(-1); }
8207    if (iterator==0) {
8208      to_use_ntableitems = 1;
8209    } else {
8210      to_use_ntableitems = ntableitems;
8211    }
8212    for (i=0;i<to_use_ntableitems;i++) {
8213      save_all_coefficients_of_basis_functions();
8214
```

```
8215
compute_error_and_prediction_based_on_w0_b56_and_basis_functions(n_examples_i
n_file);
8216     fill_index_of_examples_descending_abs_error(n_examples_in_file);
8217
8218
compute_index_with_max_abs_error_and_max_abs_error(n_examples_in_file,&index_
with_max_abs_error,&max_abs_error);
8219
8220     printf("    max_abs_error = %21.18lf\n",max_abs_error);
8221
8222     old_max_abs_error = max_abs_error;
8223
fill_derivative_of_loss_wrt_k_considering_minimize_max_abs_error_from_samples
(tableitems[i].nsamples); // this relies on index
8224     print_derivative_of_loss_wrt_k_to_file_iterator_and_i(iterator,i);
8225     alpha = tableitems[i].alpha;
8226     multiply_alpha(iterator,&alpha);
8227     change_all_coefficients_of_basis_functions(alpha);
8228
8229     printf("    Start i=%d alpha = %21.18lf index_with_max_abs_error =
%d max_abs_error =
%21.18lf\n",i,alpha,index_with_max_abs_error,max_abs_error);
8230
8231
compute_error_and_prediction_based_on_w0_b56_and_basis_functions(n_examples_i
n_file);
8232     fill_index_of_examples_descending_abs_error(n_examples_in_file);
8233
compute_index_with_max_abs_error_and_max_abs_error(n_examples_in_file,&index_
with_max_abs_error,&max_abs_error);
8234
8235     while ((max_abs_error>old_max_abs_error) &&
(max_abs_error>0.000000000000001)) {
8236       printf("    Begin loop i=%d alpha = %21.18lf
index_with_max_abs_error = %d max_abs_error =
%21.18lf\n",i,alpha,index_with_max_abs_error,max_abs_error);
8237
8238       restore_all_coefficients_of_basis_functions();
8239       divide_alpha(iterator,&alpha);
8240       change_all_coefficients_of_basis_functions(alpha);
8241
8242
compute_error_and_prediction_based_on_w0_b56_and_basis_functions(n_examples_i
n_file);
8243       fill_index_of_examples_descending_abs_error(n_examples_in_file);
8244
compute_index_with_max_abs_error_and_max_abs_error(n_examples_in_file,&index_
with_max_abs_error,&max_abs_error);
8245       printf("    End loop i=%d alpha = %21.18lf
index_with_max_abs_error = %d max_abs_error =
%21.18lf\n",i,alpha,index_with_max_abs_error,max_abs_error);
8246     }
8247     printf("    i=%d alpha = %21.18lf index_with_max_abs_error = %d
max_abs_error = %21.18lf\n",i,alpha,index_with_max_abs_error,max_abs_error);
8248
8249     tableitems[i].alpha = alpha;
```

```
8250      tableitems[i].max_abs_error = max_abs_error;
8251      print_basis_functions_to_file(tableitems[i].fn_basis_coefficients);
8252
8253      restore_all_coefficients_of_basis_functions();
8254    }
8255    if (iterator==0) {
8256      copy_alpha_from_first_tableitem_to_others();
8257      make_all_except_first_tableitem_very_high_max_abs_error(); // since
we have only computed maxabserror for i=0, we want to make sure that we don't
choose an i>0;
8258                                                                 // thus,
we give them very large maxabserror
8259    }
8260    free(array_with_indices_for_sorting_descending_abserror);
8261  }
8262
8263  int get_best_tableitem() {
8264    int i; int i_best;
8265    i_best = 0;
8266    for (i=1;i<ntableitems;i++) {
8267      if (tableitems[i_best].max_abs_error>tableitems[i].max_abs_error) {
8268        i_best = i;
8269      }
8270    }
8271    return i_best;
8272  }
8273
8274  /*
8275  void
train_model_with_basis_functions_adjust_w5_minimize_max_abs_error(char*
fn_prefix,int n_examples_in_file) {
8276    int iterator; int best_i;
8277    initialize_tableitems(fn_prefix,n_examples_in_file);
8278    print_tableitems();
8279    for (iterator=0;iterator<10;iterator++) {
8280      printf("iterator = %d\n",iterator);
8281      fill_tableitems(n_examples_in_file,iterator);
8282      best_i = get_best_tableitem();
8283      printf("    best_i = %d\n",                                  best_i
);
8284      printf("    tableitems[best_i].alpha = %21.18lf\n",
tableitems[best_i].alpha                 );
8285      printf("    tableitems[best_i].max_abs_error = %21.18lf\n",
tableitems[best_i].max_abs_error         );
8286      printf("    tableitems[best_i].fn_basis_coefficients =
%s\n",tableitems[best_i].fn_basis_coefficients);
8287
read_basis_functions_from_file(tableitems[best_i].fn_basis_coefficients);
8288    }
8289  }
8290  */
8291
8292  /*
8293  double get_mul_from_iterator2(int iterator2) {
8294    if (iterator2==0) {
8295      return 1.0;
8296    } else {
```

```
8297      if (iterator2==1) {
8298        return 0.1;
8299      } else {
8300        return 0.01;
8301      }
8302    }
8303  }
8304  */
8305
8306  double get_mul_from_iterator2(int iterator2) {
8307    if        (iterator2==0) {
8308      return 100.00;
8309    } else if (iterator2==1) {
8310      return  10.00;
8311    } else if (iterator2==2) {
8312      return   1.00;
8313    } else if (iterator2==3) {
8314      return   0.10;
8315    } else if (iterator2==4) {
8316      return   0.01;
8317    } else {
8318      printf("*******************************This is
odd*******************************\n"); fflush(stdout);
8319      return 0.01;
8320    }
8321  }
8322
8323  void save_all_predictions_and_errors(int n_examples_in_file) {
8324    write_all_predictions_to_file("temp",n_examples_in_file);
8325    write_all_errors_to_file("temp",n_examples_in_file);
8326  }
8327
8328  void restore_all_predictions_and_errors(int n_examples_in_file) {
8329    read_all_predictions_from_file("temp",n_examples_in_file);
8330    read_all_errors_from_file("temp",n_examples_in_file);
8331  }
8332
8333  void save_array_with_indices_for_sorting_descending_abserror(int
n_examples) {
8334    char fn[200]; FILE* f; int i;
8335    sprintf(fn,"array_with_indices_for_sorting_descending_abserror");
8336    f = fopen(fn, "w+");
8337    if (f==NULL) { printf("In
save_array_with_indices_for_sorting_descending_abserror. Opening file.
failed\n"); exit(-1); }
8338    for (i=0;i<n_examples;i++) {
8339
fwrite(&(array_with_indices_for_sorting_descending_abserror[i]),sizeof(int),1
,f);
8340    }
8341    fclose( f);
8342  }
8343
8344  void restore_array_with_indices_for_sorting_descending_abserror(int
n_examples) {
8345    char fn[200]; FILE* f; int i;
8346    sprintf(fn,"array_with_indices_for_sorting_descending_abserror");
```

```
8347    f = fopen(fn, "r");
8348    if (f==NULL) { printf("In
restore_array_with_indices_for_sorting_descending_abserror. Opening file.
failed\n"); exit(-1); }
8349    for (i=0;i<n_examples;i++) {
8350
fread(&(array_with_indices_for_sorting_descending_abserror[i]),sizeof(int),1,
f);
8351    }
8352    fclose( f);
8353 }
8354
8355 void print_basis_functions_to_file_iterator_nsamples_iterator2(int
iterator,int nsamples,int iterator2) {
8356    char fn[400];
8357
sprintf(fn,"basis_functions_iterator_nsamples_iterator2_%d_%d_%d",iterator,ns
amples,iterator2);
8358    print_basis_functions_to_file(fn);
8359 }
8360
8361 void
train_model_with_basis_functions_adjust_w5_minimize_max_abs_error(char*
fn_prefix,int n_examples_in_file) {
8362    int iterator; int nsamples; int iterator2; double mul; double alpha;
double old_max_abs_error; double max_abs_error; int successful_step;
8363    array_with_indices_for_sorting_descending_abserror =
malloc(n_examples_in_file*sizeof(int)); if
(array_with_indices_for_sorting_descending_abserror==NULL) { printf("Error in
train_model_with_basis_functions_adjust_w5_minimize_max_abs_error\n"); exit(-
1); }
8364    // we could call
compute_error_and_prediction_based_on_w0_b56_and_basis_functions but this has
already been done so we don't need it
8365    //
compute_error_and_prediction_based_on_w0_b56_and_basis_functions(n_examples_i
n_file);
8366    fill_index_of_examples_descending_abs_error(n_examples_in_file);
8367
8368    save_all_coefficients_of_basis_functions();
8369    save_all_predictions_and_errors(n_examples_in_file);
8370
save_array_with_indices_for_sorting_descending_abserror(n_examples_in_file);
8371    old_max_abs_error =
fabs(manyruns[array_with_indices_for_sorting_descending_abserror[0]].error);
8372    // for (iterator=0;iterator<2;iterator++) {
8373    // for (iterator=0;iterator<4;iterator++) {
8374    for (iterator=0;iterator<8;iterator++) {
8375      // for (nsamples=1;nsamples<20;nsamples++) {
8376      for (nsamples=1;nsamples<15;nsamples++) {
8377        successful_step = 0;
8378        for (iterator2=0;(iterator2<5) && (!successful_step);iterator2++)
{
8379          mul = get_mul_from_iterator2(iterator2);
8380          alpha =
mul*get_span_of_nsamples_largest_abs_error(nsamples+1)/NBASISFUNCTIONS; //
```

```
this will give us approximately alpha := mul * 1 * 10^{-6} / ( 10 * 10^{3}) =
mul * 0.1 * 10^{-9}
8381            // hence, we get alpha in {0.1 * 10^{-9}, 0.1 * 0.1 * 10^{-9},
0.01 * 0.1 * 10^{-9}}
8382          printf("    Trying with mul=%21.18lf
alpha=%21.18lf\n",mul,alpha);
8383
fill_derivative_of_loss_wrt_k_considering_minimize_max_abs_error_from_samples
(nsamples); // this relies on index
8384
print_derivative_of_loss_wrt_k_to_file_iterator_and_nsamples_and_iterator2(it
erator,nsamples,iterator2);
8385          change_all_coefficients_of_basis_functions(alpha);
8386
print_basis_functions_to_file_iterator_nsamples_iterator2(iterator,nsamples,i
terator2);
8387
8388
compute_error_and_prediction_based_on_w0_b56_and_basis_functions(n_examples_i
n_file);
8389          fill_index_of_examples_descending_abs_error(n_examples_in_file);
8390          max_abs_error =
fabs(manyruns[array_with_indices_for_sorting_descending_abserror[0]].error);
8391
8392          if (max_abs_error<old_max_abs_error) {
8393            printf("    Successful step. iterator=%d nsamples=%d
iterator2=%d max_abs_error=%21.18lf
old_max_abs_error=%21.18lf\n",iterator,nsamples,iterator2,max_abs_error,old_m
ax_abs_error);
8394            save_all_coefficients_of_basis_functions();
8395            save_all_predictions_and_errors(n_examples_in_file);
8396
save_array_with_indices_for_sorting_descending_abserror(n_examples_in_file);
8397            old_max_abs_error =
fabs(manyruns[array_with_indices_for_sorting_descending_abserror[0]].error);
8398            successful_step = 1;
8399          } else {
8400            printf("    Failed step. iterator=%d nsamples=%d iterator2=%d
max_abs_error=%21.18lf
old_max_abs_error=%21.18lf\n",iterator,nsamples,iterator2,max_abs_error,old_m
ax_abs_error);
8401            restore_all_coefficients_of_basis_functions();
8402            restore_all_predictions_and_errors(n_examples_in_file);
8403
restore_array_with_indices_for_sorting_descending_abserror(n_examples_in_file
);
8404          }
8405        }
8406      }
8407    }
8408    free(array_with_indices_for_sorting_descending_abserror);
8409 }
8410
8411 void find_WCET_v5(char* fn_with_executiontimemeasurements_inputs,char*
fn_with_executiontimemeasurements_times,char* fn_prefix,int programid,int
n_examples_in_file) {
8412    char fn[2000]; char fn2[2000]; double t0; double t1; FILE* f;
```

```
8413    char temp1_fn_prefix[200];
8414    char temp2_fn_prefix[200];
8415    int iterator;
8416    int status_clock_gettime_begin; int status_clock_gettime_end; struct
timespec mybegin; struct timespec myend;
8417    double best_alpha; double best_max_abs_error; double
current_max_abs_error; double alpha;
8418
allocate_memory_for_predictedworstcaseinput_phase0_and_predictedworstcaseinpu
t_phase1();
8419    allocate_memory_for_learnable_weights_v2(); // this allocates memory
for w0 and b56 which we need; we don't need the others that are allocated by
this func
8420    allocate_memory_for_manyruns_v2(n_examples_in_file);
8421
read_captured_data_from_disk_two_explicit_fn(fn_with_executiontimemeasurement
s_inputs,fn_with_executiontimemeasurements_times,n_examples_in_file);
8422
calculate_min_max_average_executiontime_from_manyruns(n_examples_in_file);
8423    printf("n_examples_in_file = %d\n",n_examples_in_file);
fflush(stdout);
8424    printf("min_observed_time = %lf\n",min_observed_time);
fflush(stdout);
8425    printf("max_observed_time = %lf\n",max_observed_time);
fflush(stdout);
8426    sprintf(temp1_fn_prefix,"%s_for_affine_model_1",fn_prefix);
8427    read_w0_b56_from_textfile(temp1_fn_prefix);
8428
8429    // compute_error_and_prediction_based_on_w0_b56(n_examples_in_file);
8430    // (*p_b56_value) = (*p_b56_value) -
compute_average_error_without_weight(n_examples_in_file);
8431
8432    //
train_affine_model_initialize_w0_and_b56_use_weighting_with_buckets(n_example
s_in_file);
8433    // for (iterator=0;iterator<80;iterator++) {
8434    //
train_affine_model_adjust_w0_use_weighting_with_buckets(n_examples_in_file);
8435    // }
8436
8437    sprintf(temp1_fn_prefix,"%s_for_affine_model_1",fn_prefix);
8438    compute_error_and_prediction_based_on_w0_b56(n_examples_in_file);
8439    write_all_predictions_to_file(temp1_fn_prefix,n_examples_in_file);
8440    write_all_errors_to_file(temp1_fn_prefix,n_examples_in_file);
8441
write_summary_of_predictions_and_errors_to_files(temp1_fn_prefix,n_examples_i
n_file);
8442
8443    write_w0_b56_to_textfile(temp1_fn_prefix);
8444    read_w0_b56_from_textfile(temp1_fn_prefix);
8445
find_WCET_input_from_prediction_model_w0_b56(temp1_fn_prefix,predictedworstca
seinput,&predicted_time_predictedworstcaseinput);
8446
copy_inputtoprogram(predictedworstcaseinput_phase0,predictedworstcaseinput);
8447    predicted_time_predictedworstcaseinput_phase0 =
predicted_time_predictedworstcaseinput;
```

```
8448
8449    // generate_basis_functions_without_coefficients();
8450    //
generate_basis_functions_without_coefficients_select_among_all_pairs(fn_prefi
x,n_examples_in_file);
8451
generate_basis_functions_without_coefficients_select_among_all_pairs_temp(fn_
prefix,n_examples_in_file);
8452
sprintf(temp2_fn_prefix,"%s_for_affine_model_with_basis_functions_2",fn_prefi
x);
8453    // read_basis_functions_from_file(temp2_fn_prefix);
8454    // set_all_coefficients_of_basis_functions_to_zero();
8455
8456    // train_model_with_basis_functions_adjust_w5(n_examples_in_file);
8457    // train_model_with_basis_functions_adjust_w5(n_examples_in_file);
8458
8459    status_clock_gettime_begin = clock_gettime(CLOCK_MONOTONIC_RAW,
&mybegin); if (status_clock_gettime_begin!=0) { printf("Error in
run_program_with_input_dont_use_filesystem.
status_clock_gettime_begin=%d\n",status_clock_gettime_begin); exit(-1); }
8460
train_model_with_basis_functions_adjust_w5_minimize_max_abs_error(temp2_fn_pr
efix,n_examples_in_file);
8461    /*
8462    for (iterator=0;iterator<10;iterator++) {
8463      save_all_coefficients_of_basis_functions();
8464      best_max_abs_error = -1.0;
8465      best_alpha = -1.0;
8466      current_max_abs_error = -1.0;
8467      alpha = 0.1;
8468      while (alpha>0.000000001) {
8469
train_model_with_basis_functions_adjust_w5_use_weighting_with_buckets_horizon
tal_scan(n_examples_in_file,alpha);
8470
compute_error_and_prediction_based_on_w0_b56_and_basis_functions(n_examples_i
n_file);
8471        current_max_abs_error = compute_max_abs_error(n_examples_in_file);
8472        if (best_alpha==-1.0) {
8473          best_max_abs_error = current_max_abs_error;
8474          best_alpha = alpha;
8475        } else {
8476           if (best_max_abs_error>current_max_abs_error) {
8477              best_max_abs_error = current_max_abs_error;
8478              best_alpha = alpha;
8479           }
8480        }
8481        alpha = alpha / 2.0;
8482        restore_all_coefficients_of_basis_functions();
8483      }
8484      printf("iteration=%d best_max_abs_error=%25.21lf
best_alpha=%25.21lf\n",iterator,best_max_abs_error,best_alpha);
8485      alpha = best_alpha;
8486
train_model_with_basis_functions_adjust_w5_use_weighting_with_buckets_horizon
tal_scan(n_examples_in_file,alpha);
```

```
8487      print_derivative_of_loss_wrt_k_to_file(iterator);
8488      print_basisfunctions_coefficient(iterator);
8489      print_weights_for_buckets(iterator);
8490
compute_error_and_prediction_based_on_w0_b56_and_basis_functions(n_examples_i
n_file);
8491    }
8492    */
8493    status_clock_gettime_end = clock_gettime(CLOCK_MONOTONIC_RAW, &myend);
if (status_clock_gettime_end!=0) { printf("Error in
run_program_with_input_dont_use_filesystem.
status_clock_gettime_end=%d\n",status_clock_gettime_end); exit(-1); }
8494    //
write_single_double_to_file(fn_prefix,"time_for_train_affine_model_adjust_w5_
use_weighting_with_buckets.txt",diff_s_timespec(&myend, &mybegin));
8495    //
write_single_double_to_file(fn_prefix,"time_for_train_model_with_basis_functi
ons_adjust_w5_use_weighting_with_buckets.txt",diff_s_timespec(&myend,
&mybegin));
8496
write_single_double_to_file(fn_prefix,"time_for_train_model_with_basis_functi
ons_adjust_w5_minimize_max_abs_error.txt",diff_s_timespec(&myend, &mybegin));
8497
8498
sprintf(temp2_fn_prefix,"%s_for_affine_model_with_basis_functions_2",fn_prefi
x);
8499
compute_error_and_prediction_based_on_w0_b56_and_basis_functions(n_examples_i
n_file);
8500    write_all_predictions_to_file(temp2_fn_prefix,n_examples_in_file);
8501    write_all_errors_to_file(temp2_fn_prefix,n_examples_in_file);
8502
write_summary_of_predictions_and_errors_to_files(temp2_fn_prefix,n_examples_i
n_file);
8503
8504    print_basis_functions_to_file(temp2_fn_prefix);
8505    read_basis_functions_from_file(temp2_fn_prefix);
8506
find_WCET_input_from_prediction_model_w0_b56_w5(temp2_fn_prefix,predictedwors
tcaseinput,&predicted_time_predictedworstcaseinput);
8507
copy_inputtoprogram(predictedworstcaseinput_phase1,predictedworstcaseinput);
8508    predicted_time_predictedworstcaseinput_phase1 =
predicted_time_predictedworstcaseinput;
8509
8510    printf("Running with obtained worst-case input\n");
8511    sleep(10);
8512    setprocessoraffinity_to_allow_just_a_single_processor_proc0();
8513    sleep(60);
8514
run_program_with_two_different_inputs_dont_use_filesystem_run_X_times(predict
edworstcaseinput_phase0,predictedworstcaseinput_phase1,programid,100,&t0,&t1)
;
8515
8516    sprintf(fn, "%s_0_predictedworstcaseinput.dat",        fn_prefix);
8517    sprintf(fn2,"%s_0_predictedworstcaseexecutiontime.dat",fn_prefix);
```

```
8518
write_input_and_execution_time_to_files(fn,fn2,predictedworstcaseinput_phase0
,t0);
8519
sprintf(fn,"%s_0_predictedworstcaseexecutiontime_notrun.dat",fn_prefix); f =
fopen(fn, "w+"); fprintf( f, "%18.12lf\n",
predicted_time_predictedworstcaseinput_phase0); fclose( f);
8520
8521   sprintf(fn, "%s_1_predictedworstcaseinput.dat",        fn_prefix);
8522   sprintf(fn2,"%s_1_predictedworstcaseexecutiontime.dat",fn_prefix);
8523
write_input_and_execution_time_to_files(fn,fn2,predictedworstcaseinput_phase1
,t1);
8524
sprintf(fn,"%s_1_predictedworstcaseexecutiontime_notrun.dat",fn_prefix); f =
fopen(fn, "w+"); fprintf( f, "%18.12lf\n",
predicted_time_predictedworstcaseinput_phase1); fclose( f);
8525
8526   setprocessoraffinity_to_allow_all_processors(); // we don't need to
restore this but it does not hurt
8527
8528   free_memory_for_manyruns_v2(n_examples_in_file);
8529   free_memory_for_learnable_weights_v2(); // we only need w0 and b56
8530
free_memory_for_predictedworstcaseinput_phase0_and_predictedworstcaseinput_ph
ase1();
8531 }
8532
8533 int main(int argc, char** argv) {
8534   char fn_with_executiontimemeasurements_inputs[2000];
8535   char fn_with_executiontimemeasurements_times[2000];
8536   char fn_prefix[2000];
8537   int programid;
8538   int n_examples_in_file;
8539   int n_examples_selected;
8540   srandom(142137);
8541   srand48(142137);
8542
8543   // testing_get_row_and_col_routines();
8544   // exit(-1);
8545
8546   n_examples_in_file = 30000000;
8547   programid = PROGRAM_ID_BUBBLESORT;
8548
sprintf(fn_with_executiontimemeasurements_inputs,"/home/ba/find_wcet/steepest
_ascent_find_wcet/t12/bubblesort_twobits_30000000_all_inputs.dat");
8549
sprintf(fn_with_executiontimemeasurements_times,"/home/ba/find_wcet/steepest_
ascent_find_wcet/t12/bubblesort_twobits_30000000_all_ts.dat");
8550   sprintf(fn_prefix,"bubblesort_nn2");
8551
find_WCET_v5(fn_with_executiontimemeasurements_inputs,fn_with_executiontimeme
asurements_times,fn_prefix,programid,n_examples_in_file);
8552
8553   exit(-1);
8554
8555   n_examples_in_file = 30000000;
```

```
8556   programid = PROGRAM_ID_HEAPSORT;
8557
sprintf(fn_with_executiontimemeasurements_inputs,"/home/ba/find_wcet/steepest
_ascent_find_wcet/t13/heapsort_twobits_30000000_all_inputs.dat");
8558
sprintf(fn_with_executiontimemeasurements_times,"/home/ba/find_wcet/steepest_
ascent_find_wcet/t13/heapsort_twobits_30000000_all_ts.dat");
8559   sprintf(fn_prefix,"heapsort_nn2");
8560
find_WCET_v5(fn_with_executiontimemeasurements_inputs,fn_with_executiontimeme
asurements_times,fn_prefix,programid,n_examples_in_file);
8561
8562   n_examples_in_file = 30000000;
8563   programid = PROGRAM_ID_QUICKSORT;
8564
sprintf(fn_with_executiontimemeasurements_inputs,"/home/ba/find_wcet/steepest
_ascent_find_wcet/t16/quicksort_twobits_30000000_all_inputs.dat");
8565
sprintf(fn_with_executiontimemeasurements_times,"/home/ba/find_wcet/steepest_
ascent_find_wcet/t16/quicksort_twobits_30000000_all_ts.dat");
8566   sprintf(fn_prefix,"quicksort_nn2");
8567
find_WCET_v5(fn_with_executiontimemeasurements_inputs,fn_with_executiontimeme
asurements_times,fn_prefix,programid,n_examples_in_file);
8568
8569   n_examples_in_file = 30000000;
8570   programid = PROGRAM_ID_QSORT;
8571
sprintf(fn_with_executiontimemeasurements_inputs,"/home/ba/find_wcet/steepest
_ascent_find_wcet/t15/qsort_twobits_30000000_all_inputs.dat");
8572
sprintf(fn_with_executiontimemeasurements_times,"/home/ba/find_wcet/steepest_
ascent_find_wcet/t15/qsort_twobits_30000000_all_ts.dat");
8573   sprintf(fn_prefix,"qsort_nn2");
8574
find_WCET_v5(fn_with_executiontimemeasurements_inputs,fn_with_executiontimeme
asurements_times,fn_prefix,programid,n_examples_in_file);
8575
8576   n_examples_in_file = 30000000;
8577   programid = PROGRAM_ID_MERGESORT;
8578
sprintf(fn_with_executiontimemeasurements_inputs,"/home/ba/find_wcet/steepest
_ascent_find_wcet/t14/mergesort_twobits_30000000_all_inputs.dat");
8579
sprintf(fn_with_executiontimemeasurements_times,"/home/ba/find_wcet/steepest_
ascent_find_wcet/t14/mergesort_twobits_30000000_all_ts.dat");
8580   sprintf(fn_prefix,"mergesort_nn2");
8581
find_WCET_v5(fn_with_executiontimemeasurements_inputs,fn_with_executiontimeme
asurements_times,fn_prefix,programid,n_examples_in_file);
8582
8583   n_examples_in_file = 30000000;
8584   programid = PROGRAM_ID_MATVECMUL25;
8585
sprintf(fn_with_executiontimemeasurements_inputs,"/home/ba/find_wcet/steepest
_ascent_find_wcet/t17/matvecmul25_twobits_30000000_all_inputs.dat");
```

```
8586
sprintf(fn_with_executiontimemeasurements_times,"/home/ba/find_wcet/steepest_
ascent_find_wcet/t17/matvecmul25_twobits_30000000_all_ts.dat");
8587   sprintf(fn_prefix,"matvecmul25_nn2");
8588
find_WCET_v5(fn_with_executiontimemeasurements_inputs,fn_with_executiontimeme
asurements_times,fn_prefix,programid,n_examples_in_file);
8589
8590   n_examples_in_file = 30000000;
8591   programid = PROGRAM_ID_MATVECMUL700;
8592
sprintf(fn_with_executiontimemeasurements_inputs,"/home/ba/find_wcet/steepest
_ascent_find_wcet/t18/matvecmul700_twobits_30000000_all_inputs.dat");
8593
sprintf(fn_with_executiontimemeasurements_times,"/home/ba/find_wcet/steepest_
ascent_find_wcet/t18/matvecmul700_twobits_30000000_all_ts.dat");
8594   sprintf(fn_prefix,"matvecmul700_nn2");
8595
find_WCET_v5(fn_with_executiontimemeasurements_inputs,fn_with_executiontimeme
asurements_times,fn_prefix,programid,n_examples_in_file);
8596
8597   n_examples_in_file = 30000000;
8598   programid = PROGRAM_ID_FFT_RECT;
8599
sprintf(fn_with_executiontimemeasurements_inputs,"/home/ba/find_wcet/steepest
_ascent_find_wcet/t19/fft_rect_twobits_30000000_all_inputs.dat");
8600
sprintf(fn_with_executiontimemeasurements_times,"/home/ba/find_wcet/steepest_
ascent_find_wcet/t19/fft_rect_twobits_30000000_all_ts.dat");
8601   sprintf(fn_prefix,"fft_rect_nn2");
8602
find_WCET_v5(fn_with_executiontimemeasurements_inputs,fn_with_executiontimeme
asurements_times,fn_prefix,programid,n_examples_in_file);
8603
8604   n_examples_in_file = 30000000;
8605   programid = PROGRAM_ID_FFT_POL;
8606
sprintf(fn_with_executiontimemeasurements_inputs,"/home/ba/find_wcet/steepest
_ascent_find_wcet/t20/fft_pol_twobits_30000000_all_inputs.dat");
8607
sprintf(fn_with_executiontimemeasurements_times,"/home/ba/find_wcet/steepest_
ascent_find_wcet/t20/fft_pol_twobits_30000000_all_ts.dat");
8608   sprintf(fn_prefix,"fft_pol_nn2");
8609
find_WCET_v5(fn_with_executiontimemeasurements_inputs,fn_with_executiontimeme
asurements_times,fn_prefix,programid,n_examples_in_file);
8610
8611   n_examples_in_file = 30000000;
8612   programid = PROGRAM_ID_FFT_REALONLY;
8613
sprintf(fn_with_executiontimemeasurements_inputs,"/home/ba/find_wcet/steepest
_ascent_find_wcet/t21/fft_realonly_twobits_30000000_all_inputs.dat");
8614
sprintf(fn_with_executiontimemeasurements_times,"/home/ba/find_wcet/steepest_
ascent_find_wcet/t21/fft_realonly_twobits_30000000_all_ts.dat");
8615   sprintf(fn_prefix,"fft_realonly_nn2");
```

```
8616
find_WCET_v5(fn_with_executiontimemeasurements_inputs,fn_with_executiontimeme
asurements_times,fn_prefix,programid,n_examples_in_file);
8617
8618   n_examples_in_file = 30000000;
8619   programid = PROGRAM_ID_SIMPLEX;
8620
sprintf(fn_with_executiontimemeasurements_inputs,"/home/ba/find_wcet/steepest
_ascent_find_wcet/t22/simplex_twobits_30000000_all_inputs.dat");
8621
sprintf(fn_with_executiontimemeasurements_times,"/home/ba/find_wcet/steepest_
ascent_find_wcet/t22/simplex_twobits_30000000_all_ts.dat");
8622   sprintf(fn_prefix,"simplex_nn2");
8623
find_WCET_v5(fn_with_executiontimemeasurements_inputs,fn_with_executiontimeme
asurements_times,fn_prefix,programid,n_examples_in_file);
8624 }
8625
```