OPTIMIZATION OF MAPPING ONTO A FLEXIBLE LOW-POWER ELECTRONIC FABRIC ARCHITECTURE

by

Mustafa Baz

B.S., Middle East Technical University, 2004

M.S., University of Pittsburgh, 2005

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2008

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This dissertation was presented

by

Mustafa Baz

It was defended on

July 21, 2008

and approved by

Brady Hunsaker, Adjunct Professor, Industrial Engineering Department

Alex K. Jones, Assistant Professor, Electrical and Computer Engineering Department

Bryan A. Norman, Associate Professor, Industrial Engineering Department

Jayant Rajgopal, Associate Professor, Industrial Engineering Department

Oleg Prokopyev, Assistant Professor, Industrial Engineering Department

Dissertation Co-Director: Brady Hunsaker, Adjunct Professor, Industrial Engineering Department

Dissertation Co-Director: Alex K. Jones, Assistant Professor, Electrical and Computer Engineering Department

OPTIMIZATION OF MAPPING ONTO A FLEXIBLE LOW-POWER ELECTRONIC FABRIC ARCHITECTURE

Mustafa Baz, PhD

University of Pittsburgh, 2008

A combinatorial problem that arises from a novel electronic fabric architecture designed for low-power devices such as cellular phones and palm computers is presented. We consider the problem of efficiently mapping a given data flow graph onto a particular implementation of the fabric architecture. We formulate mixed integer linear programs (MILP) and design a sliding partial MILP heuristic for this problem. We highlight the modeling and algorithmic aspects that are necessary to make the MILP formulation competitive. The sliding partial MILP heuristic is developed to generate mappings faster and to find mappings for benchmark instances that cannot be solved by the MILP formulation.

We also present a method to tune software parameters using ideas from software testing and machine learning. The method is based on the key observation that for many classes of instances, the software shows improved performance if a few critical parameters have "good" values, although which parameters are critical depends on the class of instances. Our method attempts to find good parameter values using a relatively small number of optimization trials.

Keywords: integer programming, combinatorial optimization, heuristics, post-processing, parameter tuning, machine learning, design of experiments, low power hardware, reconfigurable hardware.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

This study is dedicated to my wonderful parents Mukaddes and İbrahim Baz.

I would like to express my gratitude to my advisor, Dr. Brady Hunsaker, for his guidance and for supporting me through my doctoral studies. Without his guidance and inspiration this work would not be complete. I also would like to thank the rest of my dissertation committee, Drs. Alex K. Jones, Jayant Rajgopal, Bryan A. Norman and Oleg Prokopyev, for their invaluable suggestions and insights.

I am grateful to my friends who created a fun work environment. Among them, special thanks to Erkut Sönmez, Zeynep Erkin, Sakine Batun, Halil Bayrak, Mehmet Gökhan, Burhaneddin Sandıkçı, Mehmet Demirci, Görkem Saka, Özlem Arısoy, Pınar Yıldırım, Tuğba Özkasap, Murat Kurt, Osman Özaltın, Rob Koppenhaver, Alp Şekerci, Anıl Yılmaz, Gözde İçten, Işıl Öndeş, Nataşa Vidic, Ava Broadway and Ural Karaduman.

Most importantly, I am forever indebted to my wonderful parents Mukaddes and İbrahim Baz. I would have never finished this dissertation without their endless love, encouragement and unconditional support.

1.0 INTRODUCTION

We consider a combinatorial problem in electronic hardware design. The hardware we consider is known as a fabric architecture. Its purpose is to balance the flexibility to handle a variety of functions with the desire to use as little energy as possible. On one end of this trade-off, application-specific integrated circuit (ASIC) hardware is the most efficient in energy usage but has no flexibility: it performs a single function. Everyday examples of ASIC hardware include MPEG2 video decoder chips for DVD players and speech compression chips for mobile phones. On the other end of the spectrum, general-purpose processors such as in personal computers provide complete flexibility but use much more energy—on the order of 100 times as much. One alternative to these are field programmable gate arrays, or FPGAs. The primary benefit of FPGAs is performance: they are easier to design and manufacture than ASIC hardware and provide better performance than a general-purpose processor. FPGAs also have the benefit that they have some flexibility: they can be "reprogrammed" to perform several alternative functions. Their energy consumption is slightly better than that of general-purpose processors.

The fabric architecture we consider falls between ASIC hardware and FPGAs. It is motivated by a desire for functional flexibility similar to an FPGA but low energy consumption similar to ASIC hardware. Example applications are mobile devices such as cell phones and music players, for which energy use is critical.

A fabric is used to perform the function of an algorithm. In typical usage, this algorithm would be a bottleneck computation of an application, called the computational kernel. The rest of the application could be run by a general-purpose processor while this computational kernel is handled more efficiently by the fabric. This is based on the well-known empirical observation that in many cases roughly 90% of the execution time is spent on roughly 10% of the code.

For example, two of the benchmark instances we consider (idctrow and idctcol) represent the inverse discrete cosine transform algorithm, which is used as part of the compression of JPEG images and MPEG movies, including movies on DVD and many movie formats used on the Internet. Two other benchmark instances (coder and decoder) represent adaptive differential pulse-code modulation, which is used to encode digital sound for use in many cellular phone systems. In both these cases, portable electronic devices like cell phones and PDAs could benefit from hardware that handled these bottleneck computations with less energy consumption than a general-purpose processor. More information about the motivation for the fabric architecture can be found in [30].

The fabric can only be used for algorithms that can be represented by a directed acyclic graph in which nodes represent arithmetic and logical operators like addition and multiplication and arcs represent output values from one operator becoming input values to another operator. Such a graph is called a data flow graph. Programming the fabric requires mapping the operators and connections of the desired data flow graph onto the ALUs and interconnect of the fabric. Thus, a fabric can be used for a given data flow graph only if such a mapping is possible. The fabric can later be reprogrammed to perform the function of another data flow graph, which is why the fabric is more flexible than dedicated ASIC hardware.

In this dissertation, we consider the problem of efficiently mapping a given data flow graph onto a given fabric. We present different techniques to address this problem. This dissertation consists of three main parts. First, we formulate mixed integer linear programs (MILP) to map a data flow graph onto a fabric. We also highlight the modeling and algorithmic aspects that are necessary to make the main formulation competitive. In the second part, we design a sliding partial MILP heuristic to generate mappings. The full MILP formulation sometimes takes too much time to generate a feasible mapping, and sometimes

cannot find a feasible mapping at all. The sliding partial MILP heuristic is developed in order to generate mappings faster and to find mappings for benchmark instances that are infeasible with current row assignments. In addition, the sliding heuristic can be used as a post-processor for other approaches. Finally, we present a method that can help software users identify good parameter values for their instances. This method is partly inspired by the MILP formulation of the mapping problem since the run times are particularly important. The method is based on the key observation that for many classes of instances, the software shows improved performance if a few critical parameters have "good" values, although which parameters are critical depends on the class of instances. We used this method to tune the parameters of CPLEX, CBC and GLPK.

1.1 CONTRIBUTIONS

This dissertation considers several problems:

Mapping Problems: The mapping problem is to assign the operators in the data flow graph to ALUs of the fabric such that the logical structure of the data flow graph is preserved and the parameters of the fabric are respected.

- *Minimum Rows Mapping:* Given a fabric width, fabric interconnect design, and data flow graph to be mapped, find a mapping that uses the minimum number of rows in the fabric.
- *Feasible Mapping with Fixed Rows:* After operators are assigned to rows so that all edges go from one row to the next, assign the operators to columns so that the fabric interconnect is respected.
- *Augmented Fixed Rows:* Like Feasible Mapping with Fixed Rows, with the addition that rows of pass-gates may be inserted to create more flexibility.

Parameter Tuning Problem: The tuning problem is to identify parameter values that are good for the instances that a user wants to solve.

Below are the contributions of this dissertation:

* A mixed integer linear program for the Feasible Mapping with Fixed Rows problem:
  We have formulated a MILP for the Feasible Mapping with Fixed Rows problem. A straightforward initial formulation was not sufficient to solve the benchmark instances which are described in Section 3.4. We present the modeling and algorithmic enhancements that are necessary to make the formulation competitive. The improved model can solve all the benchmarks for cardinality 5 interconnect in less than an hour.

* A mixed integer linear program for the Minimum Rows Mapping problem:
  We have formulated a MILP for the Minimum Rows Mapping problem. The model solves 3 of the 7 benchmark instances in 10 hours. This has not been the main focus of research due to the difficulty of solving it.

* Sliding Partial MILP Heuristic for the mapping problem:
  We develop a heuristic algorithm to solve the mapping problem starting from nearly-feasible or arbitrary solutions. This algorithm is useful in several situations. When the MILP formulation takes too long to solve, we can stop at a nearly feasible solution and apply the sliding partial MILP heuristic. The full MILP instances are sometimes infeasible with current row assignments and the sliding heuristic can overcome this issue by modifying the instances. We do not make the instances easier; we add rows when needed to increase flexibility. Other researchers in our group use simulated annealing (SA) to generate mappings, but SA cannot generate feasible mappings for some of the instances. By using the sliding partial MILP heuristic as a relatively fast post-processor, our heuristic makes their approach usable.

* Automated Tuning of Optimization Software Parameters:
  We also present a method to tune software parameters using ideas from software testing and machine learning. The full method is the result of collaboration with Brady Hunsaker, Paul Brooks and Abhijit Gosavi. We use the implementation of the method—Selection Tool for Optimization Parameters (STOP [40])—to identify good parameter values for various instances, including the mapping instances. The mapping instances are, on the average, 2.06x faster when STOP's settings are used. In the computational tests of STOP, we have found that there are typically a small number of parameters that significantly change the solution time of instances. If the critical parameters (for an

instance class) have good values, then the non-critical parameters may take any value and the resulting setting will be "good". In addition, we have found that the two configurations $p32nn32$ and $p48nn16$, both using pairwise coverage and a neural network, are good choices among the different options of STOP.

1.2 OVERVIEW OF THE DISSERTATION

The dissertation is organized as follows: Chapter 2 introduces background information for the mapping problem and methodologies used in the dissertation. Chapter 3 describes the fabric design problem mathematically and provides a literature review. This chapter discusses the benchmark instances and the interconnect designs used in the computational tests. Other approaches for the mapping problem are also explained. The MILPs for the mapping problem are considered in Chapter 4. In Chapter 5, we describe the sliding partial MILP heuristic. Different alternatives are considered for the sliding heuristic and computational tests to compare the alternatives are presented. Chapter 6 describes the automated tuning of optimization software parameters. The algorithm and computational tests on various instances are presented in this chapter. Finally, Chapter 7 presents conclusions and discusses potential future research directions.

2.0 BACKGROUND

In this chapter, the fabric architecture and the methodologies we consider in the dissertation are described in detail. We provide literature reviews for MILP applications in circuit design and post processing applications. Literature reviews for the problems we consider will be discussed in detail in Sections 3.2 and 6.3 after the problems are defined more formally.

2.1 FABRIC ARCHITECTURE

In this section, the fabric architecture is described in more detail. We first define the concepts we use:

Operator: An arithmetic or logic operator such as addition, multiplication, bit shift, or logical "and". Our applications use unary, binary, and trinary operators.

Operand: An operand is the data input to an operation. In our benchmarks, each operation takes one to three operands—usually two.

Data Flow Graph: A data flow graph is a directed graph with no directed cycles (see Figure 2.1).

Arithmetic logic unit (ALU): ALU is a digital circuit that performs arithmetic and logical operations. An ALU can perform one of several operations.

Fabric Architecture: Fabric Architecture is an electronic hardware which is comprised of rows of arithmetic and logic units and a reconfigurable interconnect.

Interconnect Design: The possible connections that can be made between adjacent rows. That is, which ALUs may send output to which other ALUs. We consider several interconnect designs.

Height: The number of rows in a fabric.

Width: The number of ALUs in each row of a fabric.

Pass-Gate: Pass-gate is an operation which takes a single input and passes the input value to one or more outputs.

2.1.1 Computer-Aided Design Background

Hardware acceleration using FPGAs has become increasingly popular for computationally intensive Digital Signal Processing (DSP) applications. For a more detailed description of DSP, see [22, 30]. Unfortunately, while FPGAs have a reasonably tractable Computer-Aided Design (CAD) flow and performance, they have poor power characteristics when compared to direct ASIC fabrication. ASICs exhibit better performance and power than FPGAs, but require complex CAD and large Non-Recurring Engineering (NRE) costs.

In this dissertation, a coarse-grained, reconfigurable fabric model that exhibits ASIC-like power characteristics and FPGA-like costs and tool support is considered. The low-power fabric was designed to operate within the SuperCISC processor architecture [22].

The proposed reconfigurable fabric model is designed to mimic the computational style of Super Data Flow Graphs [22, 30]. As shown in Figure 2.2, ALUs are organized into rows within which each functional unit operates independently. The results of these ALU operations are then fed into interconnection rows constructed using multiplexers. The details of the fabric model can be found in [29].

In order to use our fabric with a given benchmark data flow graph, it is necessary to map the data flow graph onto the fabric. Such a mapping consists of an assignment of operators in the data flow graph to ALUs of the fabric such that the logical structure of the data flow graph is preserved and the parameters of the fabric are respected, particularly the width, height, and interconnect design. Generating these mappings is the primary focus of this dissertation.

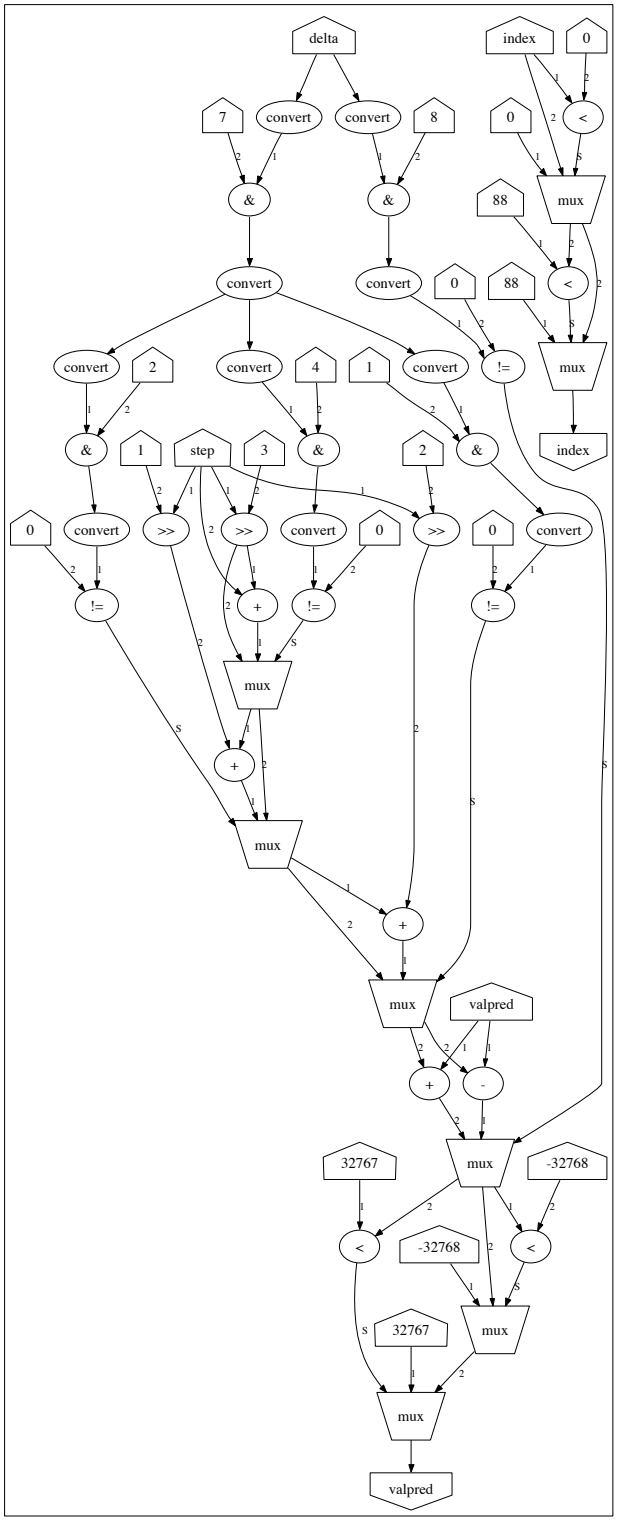


Figure 2.1: Example data flow graph.

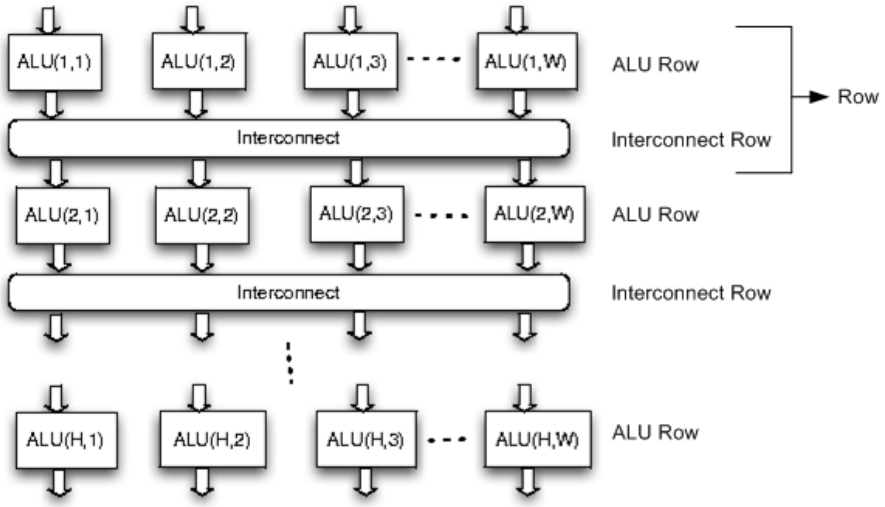


Figure 2.2: ALU row and interconnect row.

2.2 METHODOLOGIES

We formulate MILPs for the various forms of the mapping problem in Chapter 4. MILP is a common modeling and solution technique for combinatorial optimization. In contrast to linear programming (LP), which can be solved efficiently in the worst case, MILP is NP-hard. However, MILPs provide decision makers with modeling opportunities that can closely match modeling requirements for real-world systems. MILPs have been successfully applied to many planning, scheduling, etc., problems in various areas. We refer to Nemhauser and Wolsey [34] for more information on modeling and solution techniques for MILP.

In this part, we present a representative set of MILP applications in circuit design. Breuer [11] summarizes many MILP applications in the area of design automation of computers. It gives MILP formulations of problems in computer design, circuit design and computer implementation.

Goldstick and Mackie [17] have shown how MILP can be applied to the design of computer circuits, where the objective is to minimize the total power consumption. A MILP based

approach is introduced by Kumar et al. [25] to solve the layout problem; which performs placement and routing of interconnection for a given circuit schematic on a printed circuit board.

Barahona et al. [5] have implemented a standard cutting plane algorithm based on the simplex method to solve max-cut problems in circuit layout design. Niemann and Marwedel [37] describe a MILP formulation to solve the hardware/software partitioning.

The MILP applications described above consider different types of circuit problems. We are not aware of any research that has considered the mapping problem that is required for effective use of our low-energy computational fabric.

The heuristic in Chapter 5 can be used as a post-processor. Three similar applications are considered in this section: [3, 27, 42]. A post-processing heuristic is used for minimizing total trim loss in glass cutting by Arbib and Marinelli [3]. After generating a feasible solution, post-processing aims at further reducing the trim loss. The problem of minimizing the total weighted tardiness on a single machine is considered by Lee et al. [27]. In the algorithm, one of the phases is post-processing which is designed to improve the solution. Chiu-wing et al. [42] consider the optimal cell flipping problem to reduce the total wire length in a placed circuit. Cell flipping is used as a post-processing step to further reduce the total wire length. These papers use the post-processing to improve solutions. In contrast, we use a post-processing heuristic which starts from an infeasible solution and generates valid mappings by providing more flexibility. Arbib and Marinelli [3] use the same underlying methodology—MILP—as we do.

3.0 THE FABRIC DESIGN PROBLEM

A fabric consists of a number of rows of ALUs. Each ALU has a single output value and up to three inputs for operands, though most operators use only one or two operands. The input for each operand comes from ALUs only in the row immediately above, and outputs from each ALU go to ALUs only in the row immediately below. In general, an ALU does not have connections available to all the ALUs in the surrounding rows. The exact pattern of available connections is called the fabric interconnect. This fabric model is shown in Figure 2.2 on page 9. The fabric is determined by the width, the height and the interconnect pattern.

A data flow graph is a directed graph with no directed cycles. Nodes represent arithmetic/logic operators, while arcs show the connection of outputs from one operator to inputs of other operators. Nodes with no inputs represent the true "input" values of the data flow graph, while nodes with no outputs represent the output values. An example data flow graph is shown in Figure 2.1 on page 8.

The remainder of this chapter is organized as follows. Section 3.1 states the mapping problem. Section 3.2 provides a literature review. In Section 3.3, we discuss the structures in the applications which are challenging to map. Section 3.4 presents the benchmark instances used in our computational tests. Interconnect designs used in the fabric architecture are presented in Section 3.5. Other approaches used for the mapping problem are described in Section 3.6.

3.1 MAPPING PROBLEM STATEMENT

A mapping of a data flow graph onto a fabric consists of an assignment of operators in the data flow graph to ALUs of the fabric such that the logical structure of the data flow graph is preserved and the parameters of the fabric are respected. This mapping problem is central to the use of the fabric since a solution must be available each time the fabric is reprogrammed for a different data flow graph. Because of the layered nature of the fabric, the mapping is also allowed to use ALUs as "pass-gates", which take a single input and pass the input value to one or more outputs. In general, not all of the available ALUs and edges will be used. An example mapping is shown in Figure 3.1.

The interconnect design—that is, the pattern of available edges—is the primary factor in determining whether a given data flow graph can be mapped onto the fabric. For flexibility, it would make sense to provide a complete interconnect with each ALU connected to every ALU in the next row. The reason to limit the interconnect is that the cardinality of the interconnect has a significant impact on energy consumption. Although most of the connections are unused, the increased cardinality of the interconnect requires more complicating underlying hardware which leads to greater energy consumption. For a more detailed description of this phenomenon, see [30], which indicates that this energy use can be significant. Therefore, we consider limited interconnects, which have better energy consumption but make the mapping problem more challenging.

We consider the mapping problem in three forms. We call these problems Minimum Rows Mapping, Feasible Mapping with Fixed Rows and Augmented Fixed Rows. These problems are briefly described in the following subsections. Our approaches for solving the mapping problems appear in Chapters 4 and 5.

3.1.1 Minimum Rows Mapping

Given a fixed width and interconnect design, a fabric with fewer rows will use less energy than one with more rows. As data flows through the device from top to bottom it traverses ALUs and routing channels, consuming energy in each traversal. The amount of energy consumed

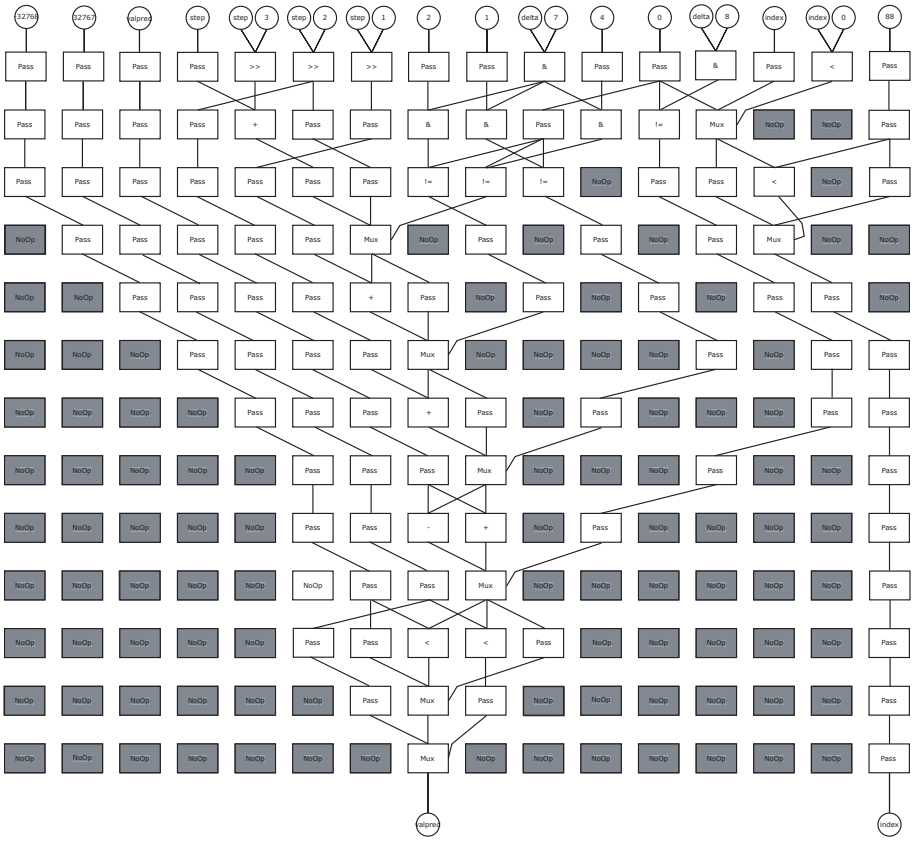


Figure 3.1: Example mapping.

varies depending on the operation that an ALU performs, however, even just passing the value through the ALU consumes a significant amount of energy. Thus, the number of rows that the data must traverse impacts how much energy is consumed. If the final result has been computed, the data can escape to an early exit bypassing the remaining rows of the fabric and reducing the energy required to complete the computation. Therefore, it is desirable to use as few rows as possible. Given a fabric width, fabric interconnect design, and data flow graph to be mapped, the Minimum Rows Mapping problem is to find a mapping that uses the minimum number of rows in the fabric. The mapping may use pass-gates as necessary.

It is easy to identify a lower bound on the number of rows by analyzing the critical path of the data flow graph. However, it is not clear whether this minimum can always be obtained for a given interconnect design.

We have formulated a MILP called IP-General in Section 4.2 for this problem. However, this problem has not been the main focus of research due to the difficulty of solving it.

3.1.2 Feasible Mapping with Fixed Rows

One of the more complicated parts of creating a mapping is the introduction of pass-gates to fit the layered structure of the fabric. One approach that we have used is to work in two stages. In the first stage, pass-gates are introduced heuristically and operators assigned to rows so that all edges go from one row to the next. The second stage assigns the operators to columns so that the fabric interconnect is respected. This second stage is called Feasible Mapping with Fixed Rows. Note that depending on the interconnect design, there may or may not exist such a feasible mapping.

We have formulated a MILP called IP-Fixed in Section 4.1 for this problem.

3.1.3 Augmented Fixed Rows

This problem first tries to solve the feasible mapping with fixed rows problem. If this is infeasible, then it may add a row of pass gates to gain flexibility. It then tries to solve Feasible Mapping with Fixed Rows on the new problem. This is repeated until a solution is found or a limit is reached on the number of rows to add.

We have developed a partial sliding MILP heuristic in Chapter 5 for this problem.

3.2 LITERATURE REVIEW

We are not aware of any research that has considered the same application for mapping. In particular, we are not aware of any previous work that relates to Minimum Rows Mapping.

There are two related problems in graph theory. First, Feasible Mapping with Fixed Rows may be viewed as a special case of subgraph isomorphism, also called subgraph containment. The data flow graph (modified to have fixed rows) may be considered as a directed graph G, and the fabric may be considered as a directed graph H. The problem is to identify an isomorphism of G with a subgraph of H.

Most of the work on subgraph isomorphism uses the idea of efficient backtracking, first presented in [41]. Examples of more recent work on the problem include [31, 13, 24]. In each of these cases, algorithms are designed to solve the problem for arbitrary graphs. In contrast, the graphs for our problem are highly structured, and our approaches take advantage of this structure. Subgraph isomorphism is NP-complete.

If we fix the number of rows in the fabric, then finding a feasible mapping (but not minimizing the number of rows) may be viewed as a special case of a problem known as directed minor containment [14, 21]. The data flow graph may be considered as a directed graph G, and the fabric may be considered as a directed graph H. Directed minor containment (also known as butterfly minor containment) is the problem of determining whether G is a directed minor of H. Unlike subgraph isomorphism, G may be a directed minor without being a subgraph; additional nodes (corresponding to "pass-gates" in our application) may

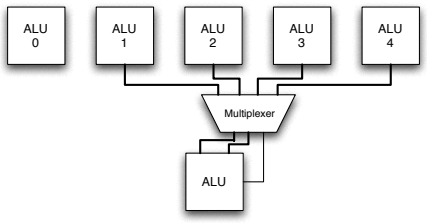


Figure 3.2: Connectivity using cardinality 4 interconnect.

be present in the subgraph of H. Directed minor containment is also NP-complete. We are not aware of any algorithms for solving directed minor containment on general graphs or graphs similar to our fabric mapping problem.

3.3 CHALLENGING STRUCTURES

As we described previously, a low-cardinality interconnect generally provides reduced energy consumption. Because of the way that the interconnect hardware works, we may restrict our attention to cardinalities that are powers of 2. So, a particular input may have 1, 2, 4, or 8 possible connections, for example.

Cardinality 1 and 2 interconnects are too small since there are nodes with three children. Based on our initial design space exploration studies, a cardinality 4 interconnect was selected to create sufficient flexibility to map our initial benchmark instances while reducing energy consumption and delay in the fabric [29, 30]. An example of this interconnection is shown in Figure 3.2. Each operand of each ALU has this same connection pattern. In this configuration, there are four possible locations to read the input operands from the previous row. The figure shows a single ALU in the lower row, but each ALU in the row will follow the same pattern.

While many structures can be successfully mapped using this interconnection pattern, a subgraph (Figure 3.3) that appears frequently in signal and image processing applications

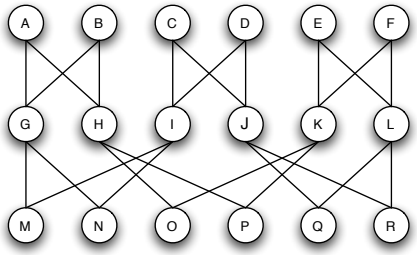


Figure 3.3: Partial data flow graph that occurs frequently in image and signal processing applications.

cannot be mapped (see [7]). To effectively map this structure, what is required is a cardinality 5 interconnect. However, a true cardinality 5 interconnect would require an additional control bit and an additional level of logic—equivalent to a cardinality 8 interconnect—which is one option we consider but is undesirable from an energy and performance perspective.

The connectivity shown in Figure 3.4 is a compromise that allows an emulation of a cardinality 5 interconnect without increasing the architectural complexity beyond cardinality 4. It does this by using a connection cardinality of 4 for each of the ALU operands, but using a different set of four inputs for each operand. Note that most operands have two inputs, which are shown in Figure 3.4 as the left and center multiplexers. The right multiplexer represents a rarely used third input. In this case, the three internal ALUs, 1–3, are shared on all operands' inputs. The outermost ALUs, 0 and 4, are only available on the left and middle/right operands, respectively. The rationale for this is that if an operand is placed to the far left or right ALU, the other operand cannot occupy the same space, thus there is no conflict for the resource. The biggest limitation to this approach is that for non-commutative operations such as subtract, there is some restriction as to which operand may be retrieved from the far left or far right. For example, in cardinality 5 interconnect inputs can come from ALUs 0-3 for the subtract operation (left operand). ALU 4 cannot be used. For the right operands inputs can come from ALUs 1-4. The mapping methods presented in this dissertation all take this non-commutativity into account.

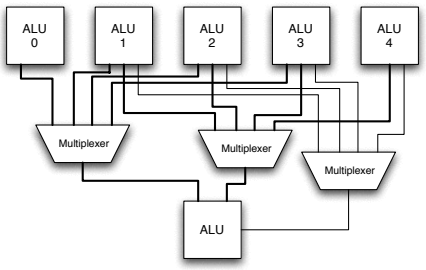


Figure 3.4: Schematic for a cardinality 5 multiplexer equivalent using cardinality 4 multiplexers

Looking back at Figure 3.3, it is clear that the mapping shown will work for our cardinality 5 interconnect. Based on our study of several data flow graphs, we have found that cardinality 5 connectivity provides a good baseline to relatively easily map.

The limitation of non-commutative operation mapping could be overcome by providing a separate operation that executes the operation right to left rather than left to right. This increases the complexity and potentially the energy consumption of each ALU since it must be capable of performing more operations. In initial experiments, we found that this "symmetric" option provides little additional mapping benefit so we have not used it.

Different interconnects have different trade-offs such as energy consumption and ease of mapping. Low cardinality interconnects provide reduced energy consumption, but it is hard to find valid mappings for these interconnects. On the other hand, high cardinality interconnects consume more energy, but they are easier to map. These trade-offs will be considered in our computational studies. The set of interconnects we consider is explained in Section 3.5.

3.4 TEST BENCHMARKS

We consider a set of seven benchmark instances from image and signal processing algorithms, which are algorithms of interest for stand-alone and low-power device implementations such as cellular phones and palm computers. All of these benchmarks except sobel [15] and laplace [16, 20] are part of the Mediabench benchmark suite [26]. Sobel and laplace are commonly available codes. They are described in more detail below:

sobel: An algorithm to find the edges between features in an image. The Sobel edge detection technique calculates these edges by computing the gradient in two directions of 3 x 3 blocks of pixels. This instance contains 51 operators, see Figure A1 on page 118.

laplace: An algorithm that computes the same information as Sobel but rather than the gradient it computes the 2^{nd} derivative in two directions of a 5 x 5 block of pixels. This instance contains 56 operators, see Figure A2 on page 119.

gsm: The core channel encoding kernel used in wireless communications for digital mobile phones. This standard is commonly used in Europe and is growing in the United States. This instance contains 106 operators, see Figure A3 on page 120.

decoder: ADPCM decoding is a channel decoding algorithm based on GSM. This instance contains 118 operators, see Figure A4 on page 121.

encoder: ADPCM encoding is the complement to ADPCM decoding. This instance contains 170 operators, see Figure A5 on page 122.

idctrow: A row-wise decomposition of a two-dimensional inverse discrete cosine transform (DCT) that was extracted from the MPEG II decoding benchmark. MPEG II is a video compression algorithm commonly used in DVD movies. This instance contains 173 operators, see Figure A6 on page 123.

idctcol: This is a column-wise decomposition of the inverse DCT. This instance contains 197 operators, see Figure A7 on page 124.

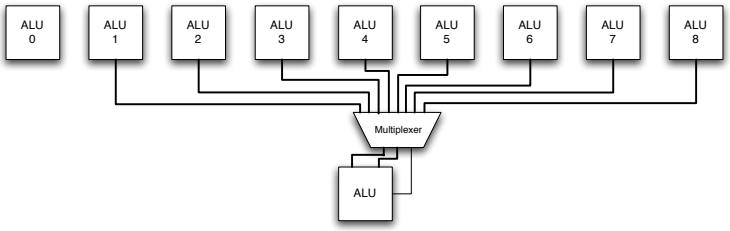


Figure 3.5: Connectivity using a cardinality 8 interconnect.

3.5 INTERCONNECT DESIGNS

The interconnect designs tested are based on cardinalities 8 and 5. Cardinality 8 interconnect uses identical interconnect patterns for each operand and each ALU. The inputs to an ALU may come from between four columns to the left and three columns to the right. The motivation is to provide relatively easy mapping with a uniform pattern. Another design for cardinality 8 interconnect is using columns between three to the left and four to the right. The choice of left vs. right was arbitrary, but we expect that it makes little if any difference. We do not use cardinality 9 interconnect using the same trick as with cardinality 5 since cardinality 8 interconnect provides sufficient flexibility to map.

Cardinality 5 interconnect actually uses separate interconnects of cardinality 4 for each operand to simulate a cardinality 5 interconnect, as discussed in Section 3.3. The motivation is to provide lower energy consumption. These two interconnect designs are shown in Figures 3.4 and 3.5.

3.5.1 Dedicated Pass-gates

When mapping a data flow graph, edges often traverse multiple rows. In these fabrics, ALUs must often pass these values through without doing any computation. We call these operations in the graph pass-gates. Figure 3.6 provides a comparison of ALUs used in the benchmark instances, showing that more than 50% of the ALUs in the fabric will be used for

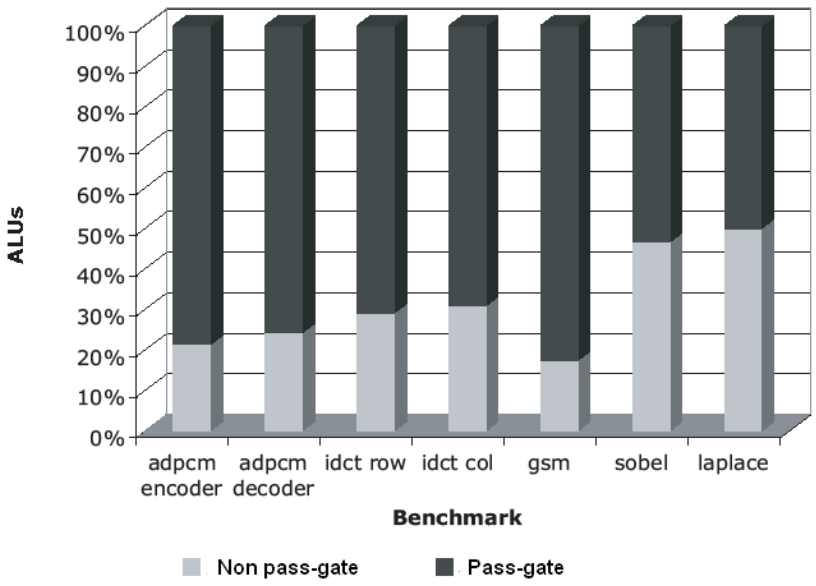


Figure 3.6: Comparison of ALUs used for routing and for computation.

routing by configuring the ALU as a pass-gate. Thus, some percentage of the ALUs in the fabric can be replaced with *dedicated* pass-gates to reduce energy consumption. The amount of energy consumed by a dedicated pass-gate is negligible compared to an ALU configured to be a pass-gate [28].

However, the introduction of these dedicated pass-gates can make it more difficult to map dense subgraphs. If too many ALUs are replaced with dedicated pass-gates, the interconnect becomes too restrictive. For example the addition of even a single pass-gate within the subgraph shown in Figure 3.3 on page 17 makes it impossible to be mapped with the cardinality 5 interconnect.

3.5.2 Heterogeneous ALUs

We also consider the impact of reducing the number of operators supported by the ALUs in the fabric. Each ALU can be programmed to handle a limited number of operations. These ALUs are called heterogeneous ALUs. The interconnects considered are 16ops-8, 10ops-8,

8ops-8, 16ops-5, 10ops-5, 16ops-88P and 10ops-88P. The first part describes the number of operations supported by each ALU and the second part is the interconnect pattern. For instance in the "10ops-8" interconnect design, each ALU can support at most 10 operations and the inputs to an ALU follow the cardinality 8 pattern. In the "16ops-88P" design, each ALU can support at most 16 operations and 33% of the ALUs are replaced by dedicated pass-gates.

As the number of operations supported by the ALUs is reduced, the complexity of the hardware is reduced which leads to energy savings. However, when the ALUs support fewer operations, the mapping problem becomes more difficult.

3.6 OTHER APPROACHES

This section discusses the other approaches—simulated annealing (SA) and a greedy heuristic—used by the researchers in our group to solve the various forms of the mapping problem. SA cannot solve some of the instances. Our sliding partial MILP heuristic, which is described in Chapter 5, makes SA usable. The greedy heuristic will be compared with the sliding partial MILP heuristic in terms of fabric size, path length and run times in Section 5.6 to evaluate the performance of the sliding heuristic.

3.6.1 Simulated Annealing

A simulated annealing algorithm [7] was developed for solving the problem of Feasible Mapping with Fixed Rows. SA is a generic probabilistic meta-algorithm for a global optimization problem, namely locating a good approximation to the global optimum of a given function in a large search space. SA is a popular algorithm for CAD flows targeting custom hardware (either for standard cell ASICs or FPGAs) particularly for placement of cells.

The simulated annealing formulation uses a graded cost function based on the multiplexing cardinality required to satisfy the edges. Edges that can be satisfied by relatively small multiplexers (from the three ALUs in the same column, one column left, or one column right)

have zero cost, edges satisfied by cardinality 5 multiplexers (up to two columns away, as in Figure 3.4) cost 1, edges requiring cardinality 8 multiplexers (as in Figure 3.5) cost 10, edges requiring cardinality 16 multiplexers (from ALUs seven columns to the left to eight columns to the right) cost 100, and edges requiring cardinality 32 multiplexers (fifteen columns to the left to sixteen columns to the right) cost 1000. Based on initial tests, this graded cost function tends to perform better than alternative cost functions as it helps to push the edges toward lower cardinality requirements, even if the edges cannot be made truly feasible yet. Subsequent iterations may then be able to improve it further.

The neighborhood for the model consists of two possible moves: (1) a node is moved from one column location to an unoccupied column within the same row or (2) two nodes within the same row are selected and swapped. A candidate move is selected uniformly at random from the choices above.

For a more detailed description of this approach, see [7].

3.6.2 Greedy Heuristic

A heuristic mapping algorithm [7, 39] was developed to solve the problem of Minimum Rows Mapping.

The algorithm works in two phases:

Row Assignment Phase: Determines the initial fabric row for each operation. Rows are determined using an as-soon-as-possible method such that operators are placed in the highest row in which they can be placed given the highest row of each of their parent operators. This phase also considers the fan-out of each operation, operations which violate the maximum fan-out, as determined by the fabric interconnect model, are fixed by pushing some operations to a lower fabric row. In addition, pass-gates are inserted into the graph such that all edges go from one row to an adjacent row.

Column Assignment Phase: Starting with the first row and continuing downward until the last, each operation in the current row is placed into an ALU location which allows the operation's edges to reach the location of each of its inputs. Once an entire row is placed the locations of each operation in the row is locked for the duration of the Mapper.

The overall structure of the algorithm is shown below.

Heuristic Mapping Algorithm:

Create initial row assignments based on as-soon-as-possible layout.

$r \leftarrow 1$

while operators are assigned to row r **do**

Assign Columns in Row r, possibly pushing some operators to later rows.

$r \leftarrow r + 1$

end while

During the assignment of operators to columns three primary factors are considered: parent dependency window, child dependency window, ALU desirability. As a result, the greedy heuristic looks ahead several rows. For a more detailed description of this approach, see [7] and [39] which explain the final algorithm.

4.0 MIXED INTEGER LINEAR PROGRAMS

In this chapter, we discuss the mixed integer linear programs we used to solve two forms of the mapping problem. We call these MILPs IP-Fixed for the feasible mapping with fixed rows problem and IP-General for the minimum rows mapping problem. The initial IP-Fixed formulation is not sufficient to solve most of the benchmark instances in the specified time limit. The modeling and algorithmic aspects that are necessary to make the formulation competitive are presented.

Other researchers in our group consider different approaches for the mapping problem. MILP solutions are important because they provide a reference for how good a mapping is possible. However, the run times for the MILPs, which will be discussed in Sections 4.1.3, 4.1.4 and 4.2.1, are quite long. In practice, the MILPs are likely to be a good choice if minimizing the number of rows is of high importance and using several hours of processing time is acceptable. Chapter 5 describes how these MILPs may be used in coordination with a post-processing heuristic to reduce solution times.

The remainder of this chapter is organized as follows. In Section 4.1, the IP-Fixed formulation for the feasible mapping with fixed rows problem and its computational results are presented. Section 4.2 describes the IP-General formulation, which is used to solve the minimum rows mapping problem. Concluding remarks are provided in Section 4.3.

4.1 IP-FIXED

We use a mixed integer linear program, which we call IP-Fixed, to solve the Feasible Mapping with Fixed Rows problem which is described in Section 3.1.2. To demonstrate the important

aspects of the formulation, an earlier formulation is presented and compared. The initial formulation is not very successful at generating mappings. Hence in the improved formulation, we explain the necessary adjustments to make the model better.

4.1.1 Initial Formulation

The input parameters and variables used in the initial formulation are given below:

Parameters:

n: Number of operators

r: Number of rows

c: Number of columns

a_i: Index of the first operator in row i

Sets:

C: Set of columns {1,. . . ,c}

R: Set of rows {1,. . . ,r}

V: Set of operators {1,. . . ,n}

E: Set of edges

E_L: Set of edges into a "left" input

E_R: Set of edges into a "right" input

E_C: Set of edges that work in either input (such as pass-gate inputs)

Variables:

x_{ij}: Binary variable for operator assignment. If operator i is in column j, then $x_{ij} = 1$, otherwise it is 0.

Formulation for cardinality 5 interconnect:

$$
\begin{array}{lrcll}
\min & & 0 & & (0) \\
s.t. & \sum_{j \in C} x_{ij} & = & 1 \quad \forall i \in V & (1) \\
& \sum_{i \in V, a_k \leq i < a_{k+1}} x_{ij} & \leq & 1 \quad \forall k \in R, j \in C & (2) \\
& x_{tk} & \leq & \sum_{j \in C, j=k-2}^{j=k+1} x_{ij} \quad \forall (i,t) \in E_L, k \in C & (3) \\
& x_{tk} & \leq & \sum_{j \in C, j=k-1}^{j=k+2} x_{ij} \quad \forall (i,t) \in E_R, k \in C & (4) \\
& x_{tk} & \leq & \sum_{j \in C, j=k-2}^{j=k+2} x_{ij} \quad \forall (i,t) \in E_C, k \in C & (5) \\
& x_{ij} & \geq & 0 \quad \forall i \in V, j \in C & (6)
\end{array}
$$

The objective function (0) is constant since the purpose is to find any feasible solution. Constraint (1) ensures that an operator can be placed in only one column. Constraint (2) states that there can be at most one operator in a column for a given row k. Constraints (3)–(5) are for the interconnect design. For example, in a cardinality 5 interconnect design, the "left" input to an ALU can have input up to the two columns to the left and one column to the right.

4.1.2 Improved Formulation

The straightforward formulation above did not do well on the benchmark instances. We found that the formulation provides weak linear programming (LP) relaxations and branching constraints based on the x variables provide little help.

A weakness of the formulation is that the edge relationships of the data flow graph are implicit in the constraints, but they cannot be branched on. The improved formulation has two main changes: new variables for edge assignments and an objective function. With these changes the MILP can solve all seven benchmark instances for cardinality 5 interconnect.

The improved formulation introduces a new binary variable $z_{i,t,j,k}$ for edge assignment. If starting operator i is in column j and ending operator t is in column k, then $z_{i,t,j,k} = 1$, otherwise it is 0. To write the formulation more easily, let parameter $p_{i,t,j,k} = 1$ if columns j and k do not have an interconnect edge for edge (i,t) (note that this depends on whether (i,t) is a "right", "left", or "center" input). Otherwise $p_{i,t,j,k} = 0$.

Below is the improved MILP formulation for cardinality 5 interconnect:

$$
\begin{array}{rrcll}
\min & \sum_{(i,t)\in E}\sum_{j\in C}\sum_{k\in C} p_{i,t,j,k} z_{i,t,j,k} & & & (0)\\
s.t. & \sum_{j\in C}\sum_{k\in C} z_{i,t,j,k} & = & 1 \quad \forall (i,t)\in E & (1)\\
 & \sum_{j\in C} x_{ij} & = & 1 \quad \forall i \in V & (2)\\
 & \sum_{i\in V, a_k \le i < a_{k+1}} x_{ij} & \le & 1 \quad \forall k \in R, j \in C & (3)\\
 & \sum_{k\in C} z_{i,t,j,k} & \le & x_{ij} \quad \forall (i,t)\in E, j \in C & (4)\\
 & \sum_{j\in C} z_{i,t,j,k} & \le & x_{tk} \quad \forall (i,t)\in E, k \in C & (5)\\
 & x_{ij} & \ge & 0 \quad \forall i \in V, j \in C & (6)\\
 & z_{i,t,j,k} & \ge & 0 \quad \forall (i,t)\in E, j \in C, k \in C & (7)
\end{array}
$$

The objective function (0) minimizes the number of the edges used that are outside the interconnect design. It basically replaces constraints (3)–(5) of the initial formulation. The cost of using each such edge is 1. If the MILP formulation finds an optimal solution with objective value zero, then the solution is a valid mapping for the given interconnect design. If the optimal objective value is greater than zero, then there is no feasible mapping for that interconnect design. The initial formulation considers a feasibility problem, by introducing an objective function we make it an optimality problem for which the MILP solvers perform better.

Constraint (1) ensures that an edge can be located in only one place. Constraints (2) and (3) are the same as constraints (1) and (2) of the initial formulation. The final two constraints relate the operator (x) and edge (z) variables. Constraint (4) states that edge (i,t) can only be placed starting at column j if operator i is at column j. Constraint (5) states that edge (i,t) can only be placed to end at column k if the ending operator t is at column k.

4.1.3 Computational Tests

To test the MILPs, we considered the seven benchmark instances described in Section 3.4. CPLEX 9.0 [19] was used to solve the resulting MILP instances on a hyper-threaded Pentium 4 operating at 3.2 GHz with 4G of memory running Linux. Run times of the two formulations for the seven benchmark instances are summarized in Table 4.1. A ten-hour time limit was

Table 4.1: Times to identify a solution using the MILP models for the cardinality 5 interconnect (seconds).

Instance	Initial formulation	Improved formulation
Sobel	5	33
Laplace	24024	46
Gsm	350	3225
Decoder	> 36000	2916
Encoder	> 36000	2746
Idctcol	> 36000	2740
Idctrow	> 36000	2211

placed on tests, and an entry of "> 36000" means that the formulation did not find a feasible solution within this time limit.

As the table indicates, the initial formulation was faster for the two instances where it could find a solution within ten hours, but for four of the instances it could not find a solution within the time limit. For "laplace" the initial model could find a solution in nearly 7 hours while the improved model solves it in 46 seconds. The improvements to the formulation allow for all seven benchmarks to be solved; the longest taking less than an hour.

4.1.4 Other MILP Formulations

This section considers the IP-Fixed formulation for interconnects with dedicated pass-gates (Section 3.5.1) and/or with heterogeneous ALUs (Section 3.5.2).

Additional sets used in the dedicated pass-gates formulation are given below:

Sets:

C_1: Set of non-dedicated columns

C_2: Set of dedicated columns

V_1: Set of operators that are not pass-gates

V_2: Set of pass-gates

A new binary variable y_{ij} for pass-gate assignment is introduced. If pass-gate i is at column j, then $y_{ij} = 1$, otherwise it is 0. To write the formulation more easily, let parameter $p_{i,t,j,k} = 100$ if columns j and k do not have an interconnect edge for edge (i, t). The reason why 100 is used will be explained after the formulation is given.

Below is the MILP formulation for cardinality 5 interconnect with dedicated pass-gates:

$$\begin{aligned}
\min \quad & \textstyle\sum_{(i,t)\in E} \sum_{j\in C} \sum_{k\in C} p_{i,t,j,k} z_{i,t,j,k} + \sum_{i\in V_2} \sum_{j\in C_1} y_{i,j} && && (0)\\
s.t. \quad & \textstyle\sum_{j\in C} \sum_{k\in C} z_{i,t,j,k} &= 1 \quad & \forall (i,t) \in E && (1)\\
& \textstyle\sum_{j\in C_1} x_{ij} &= 1 \quad & \forall i \in V_1 && (2)\\
& \textstyle\sum_{j\in C} y_{ij} &= 1 \quad & \forall i \in V_2 && (3)\\
& \textstyle\sum_{i\in V_1, a_k \le i < a_{k+1}} x_{ij} + \sum_{i\in V_2, a_k \le i < a_{k+1}} y_{ij} &\le 1 \quad & \forall k \in R, j \in C_1 && (4)\\
& \textstyle\sum_{i\in V_2, a_k \le i < a_{k+1}} y_{ij} &\le 1 \quad & \forall k \in R, j \in C_2 && (5)\\
& \textstyle\sum_{k\in C} z_{i,t,j,k} &\le x_{ij} \quad & \forall (i,t) \in E, j \in C_1 && (6)\\
& \textstyle\sum_{j\in C} z_{i,t,j,k} &\le x_{tk} \quad & \forall (i,t) \in E, k \in C_1 && (7)\\
& \textstyle\sum_{k\in C} z_{i,t,j,k} &\le y_{ij} \quad & \forall (i,t) \in E, j \in C && (8)\\
& \textstyle\sum_{j\in C} z_{i,t,j,k} &\le y_{tk} \quad & \forall (i,t) \in E, k \in C && (9)\\
& x_{ij} &\ge 0 \quad & \forall i \in V_1, j \in C_1 && (10)\\
& y_{ij} &\ge 0 \quad & \forall i \in V_2, j \in C && (11)\\
& z_{i,t,j,k} &\ge 0 \quad & \forall (i,t) \in E, j \in C, k \in C && (12)
\end{aligned}$$

The objective function (0) minimizes the sum of the number of the edges used that are outside the interconnect design and the sum of the number of pass-gates used in non-dedicated columns. The cost of using an edge outside the interconnect is 100 and the cost of using an ALU as a pass-gate is 1. There is 100:1 ratio since we never want to trade an edge outside the interconnect for an ALU used as a pass-gate. If the MILP formulation finds an optimal solution which has no edges outside the interconnect, then the solution is a valid mapping for the given interconnect design; otherwise there is no feasible mapping for that interconnect design. The formulation finds a solution with a minimum number of invalid edges; among

such solutions it finds one with a minimum number of ALUs used as pass-gates since this reduces energy consumption.

Constraints are similar to the constraints used in the improved MILP formulation. We only add the constraints that are necessary for the new variable y_{ij}. Constraint (3) ensures that a pass-gate can be placed in only one column. Constraint (4) ensures that an ALU can be used by at most one operation. Constraint (5) states that there cannot be more than one pass-gate in a dedicated column. The final two constraints—(8), (9)—relate the pass-gate (y) and edge (z) variables.

The formulation for interconnects with heterogeneous ALUs is also similar to the improved MILP formulation. The only difference is that if the ALU does not support the operation, then the operator cannot be placed in that column. The constraint $X_{ij} = 0$ is added to the formulation where i is the operation and j is the column. Suppose the first ALU in a row does not support the "+" operation, then the "+" operator cannot be placed in the first column.

4.1.5 Computational Tests

In this section, we present results of tests on interconnects with dedicated pass-gates and/or with heterogeneous ALUs. To test interconnects with dedicated pass-gates, we used the following designs: 888_8P, 88_8P, 8_8P, 555_8P, 55_8P and 5_8P. In 888_8P design, 25% (1 out of 4) of the ALUs are replaced with dedicated pass-gates, dedicated pass-gates and ALUs use cardinality 8 interconnect. In 5_8P design, 50% of the ALUs are replaced with dedicated pass-gates, ALUs use cardinality 5 interconnect and dedicated pass-gates use cardinality 8 interconnect. So in our tests, 25%, 33% and 50% dedicated pass-gates are used.

Table 4.2 shows the objective values at the end of the runs and the solution times of the seven benchmark instances for cardinality 8 interconnect with dedicated pass-gates. All of the instances have valid mappings. For "idctrow" with interconnect 8_8P, we have a valid mapping, however it may not be the optimal solution since the time limit—10 hours—is reached. For "encoder" with 888_8P, the optimal objective value is 61 which means 61 ALUs are used as pass-gates.

Table 4.2: Tests on cardinality 8 interconnect with dedicated pass-gates.

	888_8P		88_8P		8_8P	
Benchmarks	Obj.	Time	Obj.	Time	Obj.	Time
Gsm	49	264.08	37	664.25	0	1075.05
Encoder	61	543.9	51	550.63	9	2721.21
Decoder	24	223.09	17	464.26	0	364.58
Idctcol	37	23470.53	26	1839.77	3	4821.62
Idctrow	10	1452.23	6	641.37	1	> 36000
Sobel	0	7.92	0	32.23	0	7.93
Laplace	0	9.55	0	6.52	0	7.93

Table 4.3 summarizes the objective values and the run times of the seven benchmark instances for cardinality 5 interconnect with dedicated pass-gates. Only 9 out of 21 instances have valid mappings. Eight out of 21 instances cannot be solved within the time limit. Four instances are proven infeasible each with just one violating edge. The best integer solution for "idctcol" with interconnect 5_8P is 703 which states that there are 7 violated edges and 3 ALUs are used as pass-gates. It is hard to map cardinality 5 interconnect with dedicated pass-gates. Especially the interconnect 5_8P is too restrictive since 1 out of 2 ALUs is replaced with dedicated pass-gates and the ALUs use cardinality 5 interconnect. Only gsm can be mapped with this design.

Table 4.4 gives the optimal objective values and the run times of the instances for cardinality 8 interconnect with ALUs supporting at most 16, 10 and 8 operations. All of the instances have valid mappings for interconnects 16ops-8 and 10ops-8. The longest run time is 1739.07 seconds. 8ops-8 interconnect is harder, there are not valid mappings for "encoder" and "decoder" since the optimal objective values are greater than 100.

Tests on cardinality 5 interconnect with heterogeneous ALUs are shown in Table 4.5. Three instances cannot be solved within 10 hours. When we use the 16ops-5 interconnect,

Table 4.3: Tests on cardinality 5 interconnect with dedicated pass-gates.

	555_8P		55_8P		5_8P	
Benchmarks	Obj.	Time	Obj.	Time	Obj.	Time
Gsm	50	3709.86	37	849.79	0	1020.44
Encoder	63	> 36000	151	25067.99	309	> 36000
Decoder	25	1403.55	17	132.02	100	10055.91
Idctcol	344	> 36000	719	> 36000	703	> 36000
Idctrow	316	> 36000	308	> 36000	800	> 36000
Sobel	1	144.28	0	79.04	500	34942.24
Laplace	2	105.75	100	140.71	100	251.52

Table 4.4: Tests on cardinality 8 interconnect with heterogeneous ALUs.

Heterogeneous	16ops-8		10ops-8		8ops-8	
Benchmarks	Obj.	Time	Obj.	Time	Obj.	Time
Gsm	0	276.65	0	134.26	0	10829.8
Encoder	0	1369.73	0	276.79	1500	633.82
Decoder	0	160.29	0	169.31	2200	321.82
Idctcol	0	1739.07	0	1103.07	0	303.35
Idctrow	0	669.61	0	345.86	0	143.30
Sobel	0	108.88	0	15.85	0	12.05
Laplace	0	29.76	0	14.38	0	9.72

Table 4.5: Tests on cardinality 5 interconnect with heterogeneous ALUs.

Heterogeneous	16ops-5		10ops-5	
Benchmarks	Obj.	Time	Obj.	Time
Gsm	0	2035.19	300	791.89
Encoder	0	235.12	200	> 36000
Decoder	0	32351.50	900	4360.78
Idctcol	200	> 36000	1000	> 36000
Idctrow	0	10207.2	200	12410.3
Sobel	0	263.044	0	46.69
Laplace	0	31.41	0	12.84

there is not a valid mapping for "idctcol". 10ops-5 is even harder, only "sobel" and "laplace" have feasible mappings. When ALUs use cardinality 5 interconnect and support at most 10 operations, the design becomes too restrictive.

Table 4.6 shows the optimal objective values and the run times of the seven instances for cardinality 8 interconnect with dedicated pass-gates and heterogeneous ALUs. Only "decoder" with interconnect 10ops-88P has no feasible mapping. Other instances have valid mappings with some ALUs used as pass-gates. The longest run time is 5569.05 seconds.

Fifty-three out of 56 instances have valid mappings for cardinality 8 interconnects with dedicated pass-gates and/or with heterogeneous ALUs, however cardinality 5 interconnects with dedicated pass-gates and/or with heterogeneous ALUs are harder. More than half of the time—18 out of 35—benchmark instances do not have feasible mappings.

Table 4.6: Tests on cardinality 8 interconnect with dedicated pass-gates and heterogeneous ALUs.

Heterogeneous	16ops-88P		10ops-88P	
Benchmarks	Obj.	Time	Obj.	Time
Gsm	40	97.96	41	722.89
Encoder	58	540.47	60	1975.43
Decoder	30	208.11	929	5569.05
Idctcol	43	1023.34	40	318.88
Idctrow	27	72.88	26	78.56
Sobel	3	3.26	3	2.52
Laplace	2	6.00	2	16.64

4.2 IP-GENERAL

The next mixed integer linear program we consider is IP-General, which is used to solve the Minimum Rows Mapping problem described in Section 3.1.1.

New parameters and variables used in the formulation are given below:

Sets:

I: Set of Inputs

M: Set of Middle Operators

O: Set of Outputs

Variables:

$x_{i,j,k}$: Binary variable for operator assignment. If operator i is in row j at column k, then $x_{i,j,k} = 1$, otherwise it is 0.

z_j: Binary variable for rows. If row j is used, then $z_j = 1$, otherwise it is 0.

p_{jk}: Binary variable for pass-gates (Section 3.1). If a pass-gate is used in row j at column k, then $p_{jk} = 1$, otherwise it is 0.

Formulation for cardinality 5 interconnect:

$$
\begin{array}{lrcll}
\min & \sum_{j\in R} z_j & & & (0)\\
s.t. & \sum_{i\in V}\sum_{k\in C} x_{i,j,k} & \leq & 2cz_j \quad \forall j \in R & (1)\\
 & \sum_{i\in V} x_{i,j,k} & \leq & 1 \quad \forall j \in R, k \in C & (2)\\
 & \sum_{j\in R}\sum_{k\in C} x_{i,j,k} & \geq & 1 \quad \forall i \in M & (3)\\
 & \sum_{k\in C} x_{i,1,k} + \sum_{k\in C} x_{i,r,k} & = & 0 \quad \forall i \in M & (4)\\
 & \sum_{k\in C} x_{i,r,k} & = & 0 \quad \forall i \in I & (5)\\
 & \sum_{k\in C} x_{i,1,k} & = & 0 \quad \forall i \in O & (6)\\
 & \sum_{k\in C} x_{i,1,k} & = & 1 \quad \forall i \in I & (7)\\
 & x_{i,j,k} - \sum_{l\in C, l=k-2}^{l=k+2} x_{i,j-1,l} + p_{jk} & \leq & 1 \quad \forall i \in V, j \in R, k \in C & (8)\\
 & x_{t,j,k} - \sum_{l\in C, l=k-2}^{l=k+2} x_{i,j-1,l} - p_{jk} & \leq & 0 \quad \forall (i,t) \in E, j \in R, k \in C & (9)\\
 & x_{i,j,k} - \sum_{l\in C, l=k-2}^{l=k+2} x_{i,j-1,l} & \leq & 0 \quad \forall i \in I, j \in R, k \in C & (10)\\
 & x_{i,j,k}, z_j, p_{jk} & \geq & 0 \quad \forall i \in V, j \in R, k \in C & (11)
\end{array}
$$

The objective function (0) minimizes the number of rows used. Constraint (1) forces z_j to be 1 when there is an operator in row j. Constraint (2) states that there can be at most one operator in a column for a given row j. Constraint (3) ensures that middle operators must be placed. It is a $\geq$ constraint since the operators may have pass-gates. There cannot be any middle operator in the first and last row—Constraint (4). Constraint (5) states that inputs cannot be in the last row and similarly, constraint (6) ensures that outputs cannot be placed in the first row. Inputs must be placed in the first row, constraint (7). Constraints (8) and (9) are for the pass-gates. They state whether a pass-gate is used in row j at column k or not. Constraint (10) ensures that there are no preceding operators for the inputs.

4.2.1 Computational Tests

We used CPLEX 9.0 to solve the instances built from the IP-General formulation. Table 4.7 summarizes the run times for the seven benchmark instances. A ten-hour time limit was placed on the tests, and an entry of "> 36000" means that the formulation did not find a feasible solution within this time limit. IP-General formulation could not find a solution within the time limit for four of the instances.

Table 4.7: Time to identify a solution using the MILP models for cardinality 5 interconnect (seconds).

Instance	IP-General
Sobel	252
Laplace	1
Gsm	> 36000
Decoder	2992
Ecoder	> 36000
Idctcol	> 36000
Idctrow	> 36000

The IP-General formulation did not do well on the benchmark instances. LP relaxation of the formulation is weak and too slow. For example, it takes around 1000 seconds for idctcol. The formulation cannot find an integer solution easily. Therefore, it cannot fathom many nodes. In addition, the number of constraints and variables in the IP-General formulation are much greater than the IP-Fixed formulation. The improvement of the IP-General model has been left for future work.

4.3 CONCLUSION

This chapter considered two mixed integer linear programs—IP-Fixed and IP-General—to solve two forms of the mapping problem. Other researchers in our group consider different approaches for the mapping problem. Among these approaches, MILP solutions are important because they provide a reference for how good a mapping is possible.

The initial IP-Fixed formulation was not sufficient to solve the benchmark instances. The improved model has two main changes: new variables for edge assignments and an objective function. These modifications help the formulation on branching and strengthening the LP

relaxation. The improved model can solve all the benchmarks for cardinality 5 interconnect in less than an hour.

Interconnects with dedicated pass-gates and/or with heterogeneous ALUs are also considered. It is easier to map cardinality 8 interconnects. Only 3 out of 56 instances do not have valid mappings. However, for cardinality 5 interconnects there are not feasible mappings for 18 out of 35 instances with current row assignments. In order to map the infeasible structures, the instances should be modified; we can add rows when needed to increase flexibility. The IP-Fixed formulation cannot solve 12 out of 91 instances within the time limit—10 hours—and 11 out of 79 instances take more than 1 hour to solve. For these reasons, we design a partial sliding MILP heuristic. The heuristic, which will be discussed in Chapter 5, generates mappings faster and finds mappings for instances that cannot be solved by the MILP formulation by creating more flexibility.

The IP-General formulation for the minimum rows mapping problem solves 3 out of 7 benchmarks instances for cardinality 5 interconnect. We have left further development of this model for future work.

5.0 SLIDING PARTIAL MILP HEURISTIC

The mapping problem is central to the use of the fabric as a solution must be available in order for the fabric to be (re)programmed for a specific data flow graph. Twelve of the mapping instances cannot be solved within 10 hours even by using the improved MILP formulation. We expect the CAD community would not accept the run times for the MILP. Moreover, interconnects with dedicated pass-gates and/or with heterogeneous ALUs are infeasible for some of the instances with current row assignments. Computational results presented in Section 4.1.5 show that 21 out of 91 instances are impossible to map.

A heuristic algorithm usually guarantees to terminate finitely and sacrifices a guarantee of finding an optimal solution for an improvement in run time. We refer to Michalewicz and Fogel [32] for more information on heuristics. This chapter presents a heuristic algorithm to generate mappings faster and to find mappings for instances that cannot be solved by the IP-Fixed formulation. In addition, the heuristic algorithm can be used as a post-processor for other approaches. For example, other researchers in our group use simulated annealing to generate mappings, but simulated annealing cannot always find feasible mappings. By improving the output to create valid mappings, the sliding partial MILP heuristic makes their approach usable.

The remainder of this chapter is organized as follows. Section 5.1 is an introduction. In Section 5.2, the solution progress of the MILP instances are analyzed. Section 5.3 considers the sliding partial MILP heuristic algorithm. Different alternatives for the heuristic and computational results are discussed in Section 5.4. In Section 5.5, we describe the performance of our heuristic starting from an arbitrary solution. Section 5.6 compares our heuristic with a greedy heuristic. We complete the chapter with the concluding remarks in Section 5.7.

5.1 INTRODUCTION

We call our heuristic algorithm the Sliding Partial MILP Heuristic. It solves the mapping problem of a limited set of rows using a mixed integer linear program and creates a sliding heuristic to map the entire application. It is a "partial" MILP that models only part of the fabric. It is called "sliding" because groups of rows are selected to optimize using a top-down approach. The top-down approach works better than the bottom-up approach since there are more flexibilities at the bottom of the benchmark instances.

The sliding partial MILP heuristic can start from an arbitrary solution (stand-alone use) or can be used as a post-processor. If it is used as a post-processor, there are two main steps:

1. Run the IP-Fixed formulation, SA or any other algorithm for a while to get a good integer solution,
2. Run the heuristic to fix the violated edges.

First, we run the IP-Fixed formulation or simulated annealing on the benchmark instance to find a good integer solution—a mapping with edges outside the range of the interconnect. The violated edges of the integer solution are identified. Then, we run the sliding partial MILP heuristic which will be discussed in Section 5.3 to fix the violations. Thus, a feasible mapping is generated. When the sliding partial MILP heuristic is used in a stand-alone manner, only the second step is valid.

5.2 SOLUTION PROGRESS

One motivation for the heuristic is to generate mappings faster. The run times of the IP-Fixed formulation are long. We run the IP-Fixed formulation for some time, then we stop it and use the sliding partial MILP heuristic to identify a good solution. In order to decide how long to run the IP-Fixed model before applying the heuristic, we analyze the solution progress of the benchmarks for cardinality 5 interconnect. These can be seen in Figures 5.1, 5.2 and 5.3. Integer solutions found by CPLEX at different times are shown in these figures.

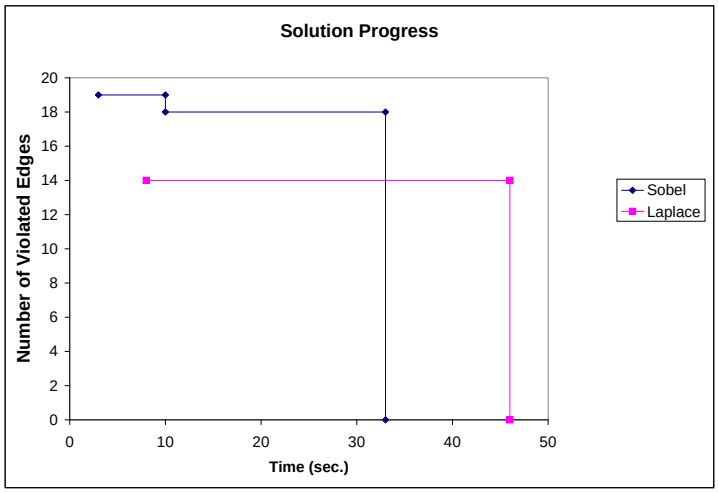


Figure 5.1: Solution progress of sobel and laplace.

The y value represents the number of violated edges in the current integer solution. When the objective value is 0, the mapping is valid.

The number of violated edges for the benchmark instances at different times can be seen in Table 5.1. "-" means an integer solution has not been found yet. By 100 seconds sobel and laplace are solved to optimality. There are 5 violated edges for gsm, 29 for decoder, 162 for encoder, 26 for idctcol and no integer solution has been found for idctrow. The IP-Fixed formulation can find good integer solutions for all the benchmarks by 600 seconds: 5 violated edges for gsm, 4 for decoder, 5 for encoder, 3 for idctcol and 9 for idctrow. There is not much improvement in the objective value at 1800 seconds; gsm and idctrow have still 5 and 9 violated edges, respectively. Therefore, 600 seconds looks like a good point to stop running the IP-Fixed model and to start running the heuristic for cardinality 5 interconnect. The IP-Fixed formulation also finds good integer solutions for other interconnects by 600 seconds.

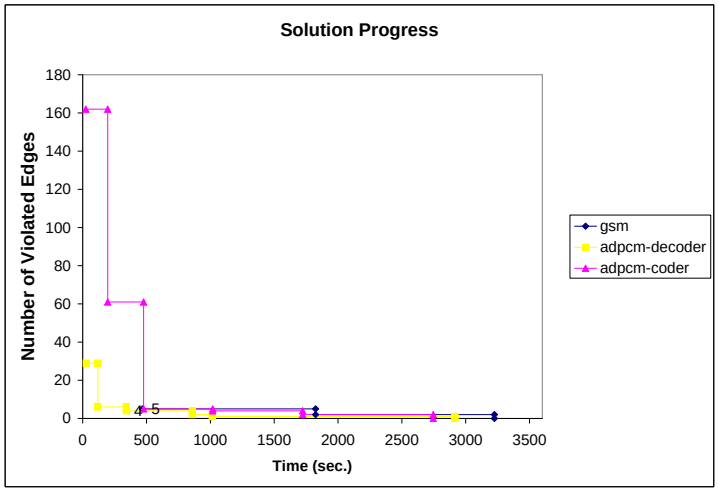


Figure 5.2: Solution progress of gsm, decoder and encoder.

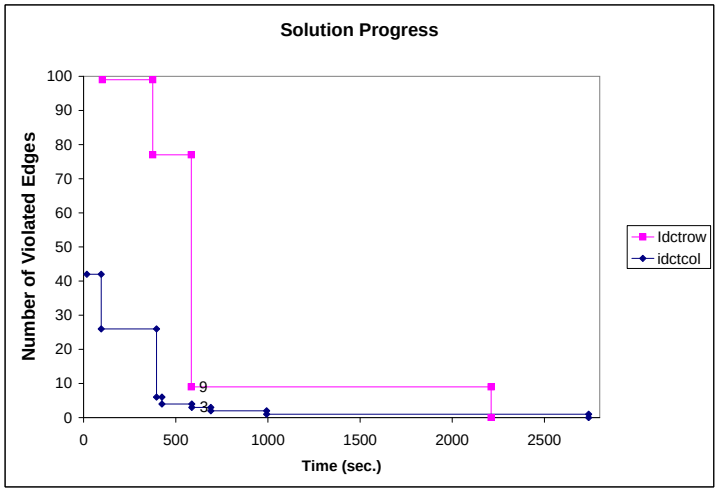


Figure 5.3: Solution progress of idctrow and idctcol.

Table 5.1: Number of violated edges at different times.

Instance	10 seconds	100 secs	300 secs	600 secs	1800 secs	3600 secs
Sobel	18	0	0	0	0	0
Laplace	14	0	0	0	0	0
Gsm	-	5	5	5	5	0
Decoder	-	29	6	4	1	0
Encoder	-	162	61	5	2	0
Idctcol	-	26	26	3	1	0
Idctrow	-	-	97	9	9	0

5.3 HEURISTIC

In this section, the sliding partial MILP heuristic algorithm is described. The heuristic generates valid mappings by using small, fast MILPs on sets of adjacent rows (Figure 5.4). The partial MILP formulation is similar to the improved IP-Fixed formulation (Section 4.1.2). Parameters and sets used in the formulation are given below:

Parameters:

$p'_{i,t,j,k}$: Objective coefficients (discussed after the formulation is given)

a_i: Index of the first operator in row i

Sets:

C: Set of columns

V': Set of operators in the MILP window (Figure 5.5)

E': Set of edges in the MILP window

R': Set of rows in the MILP window

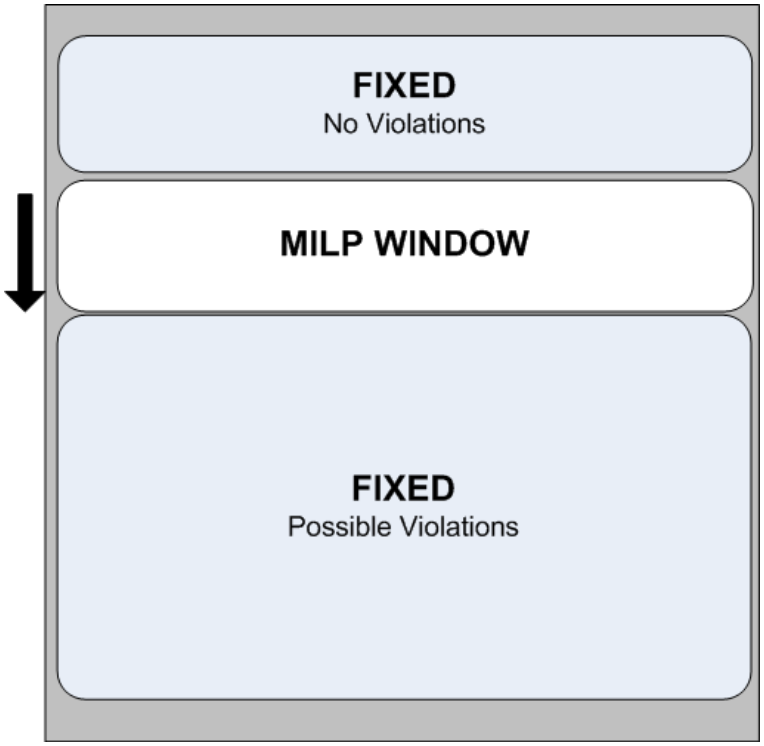


Figure 5.4: Set of adjacent rows that are selected to optimize.

Partial MILP formulation:

$$
\begin{aligned}
\min \quad & \textstyle\sum_{(i,t)\in E'} \sum_{j\in C} \sum_{k\in C} p'_{i,t,j,k} z_{i,t,j,k} && && (0)\\
s.t. \quad & \textstyle\sum_{j\in C} \sum_{k\in C} z_{i,t,j,k} &= 1 \quad & \forall (i,t) \in E' && (1)\\
& \textstyle\sum_{j\in C} x_{ij} &= 1 \quad & \forall i \in V' && (2)\\
& \textstyle\sum_{i\in V', a_k \le i < a_{k+1}} x_{ij} &\le 1 \quad & \forall k \in R', j \in C && (3)\\
& \textstyle\sum_{k\in C} z_{i,t,j,k} &\le x_{ij} \quad & \forall (i,t) \in E', j \in C && (4)\\
& \textstyle\sum_{j\in C} z_{i,t,j,k} &\le x_{tk} \quad & \forall (i,t) \in E', k \in C && (5)\\
& x_{ij} &\ge 0 \quad & \forall i \in V', j \in C && (6)\\
& z_{i,t,j,k} &\ge 0 \quad & \forall (i,t) \in E', j \in C, k \in C && (7)
\end{aligned}
$$

We restrict the locations of the operators in the first and last row of the MILP window by setting $x_{ij} = 1$ where i is the operator and j is the column.

The main idea of the sliding partial MILP heuristic is to fix the violated edges row by row. The sliding heuristic uses two ways to fix the violations: adjusting the columns of the operators and adding a row of pass-gates to create more flexibility.

Below is the general framework of the algorithm:

Sliding Partial MILP Heuristic:

while there are violated edges **do**
  Find the highest row with a violated edge.
  Solve a partial MILP to fix the violation(s), even if this creates new violations in lower rows.
  if the violation(s) is not fixed **then**
    if the number of pass-gates < limit **then**
      Add a row of pass-gates.
    else
      Exit (heuristic cannot generate a mapping).
    end if
  end if
end while

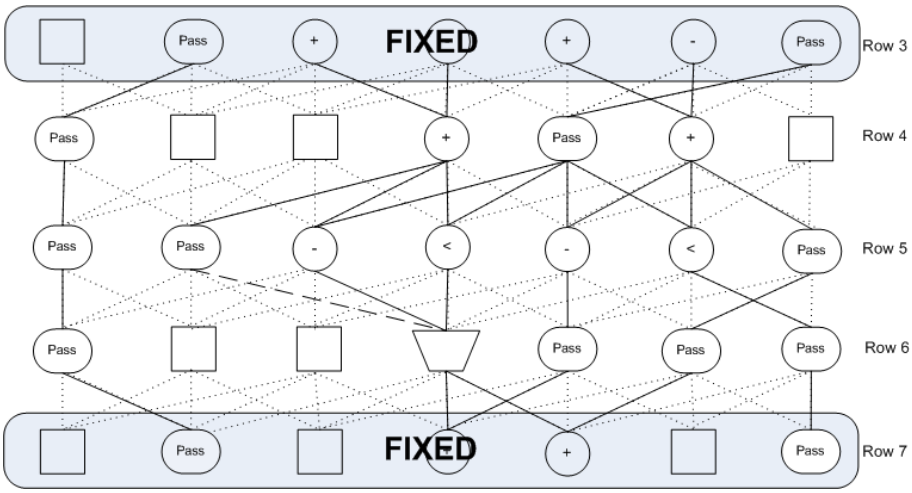


Figure 5.5: MILP window.

The heuristic continues until it fixes all the violations or adds too many rows of pass-gates, and follows a top-down approach. When it finds a violated edge, a window of rows is selected, see Figures 5.4 and 5.5, and the partial MILP adjusts column locations in this window. Because of the top-down approach there are no violations above the MILP window, but there may be violations below the window. Selection of the window of rows is discussed in Section 5.4. There are 3 possibilities after solving a partial MILP:

Case 1: The partial MILP fixes all violations in the window.

Case 2: The partial MILP fixes violations in the top row, but new or existing violations appear in lower rows of the window, see Figure 5.6. Informally, we call this "pushing the violations down in the window".

Case 3: No feasible solution can be found without violations in the top row. In this case, a row of pass-gates is added to increase the flexibility (Figure 5.7) and the partial MILP is run again.

In the partial MILP formulation, if all the violations cannot be fixed, we try to push them down to lower rows. This can be achieved with proper objective coefficients ($p'_{i,t,j,k}$) in the formulation. The objective coefficients for the violations in the upper rows are much

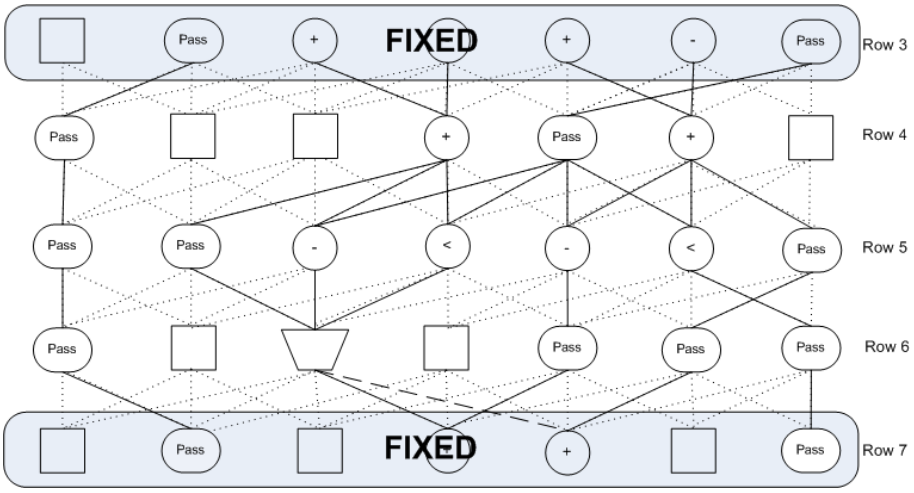


Figure 5.6: Violation is pushed down.

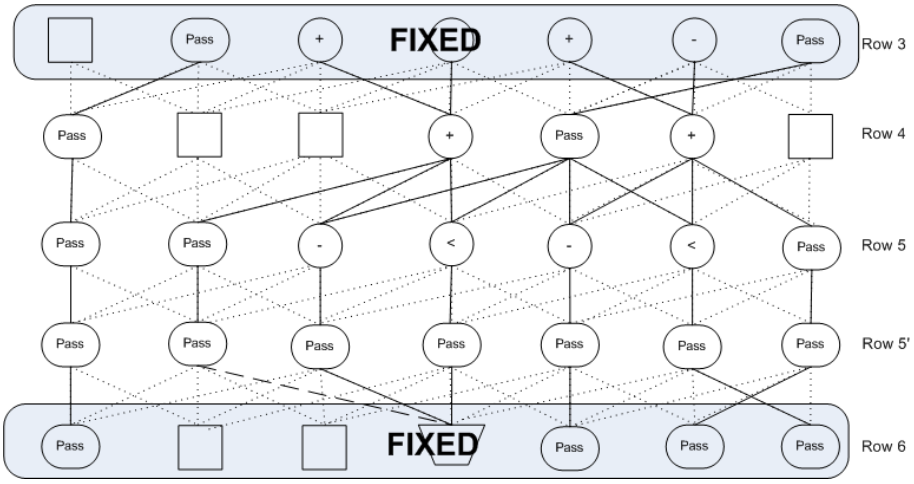


Figure 5.7: Row of pass-gates is added.

higher than the ones in the lower rows. For example, assume there is a violation between rows 5 and 6, and 3 rows are optimized, see Figure 5.5. In other words, the columns of the operators in rows 4, 5 and 6 can be adjusted and the locations of the operators in rows 3 and 7 are restricted to their current values. The objective coefficients for the edges outside the interconnect is 10000 for the edges between rows 3 and 4, also 10000 for the edges between rows 4 and 5, 100 for the ones between rows 5 and 6, and 1 for the ones between rows 6 and 7. This avoids violations in the higher rows whenever possible and may push the violation down to later rows. We also tried to move the violations to the upper rows, but in preliminary tests, this approach was not as successful as pushing them down. Test instances tend to have more operators in upper rows and so are more constrained there, which may explain this.

When the sliding partial MILP heuristic does not add any rows of pass-gates, we know that the solution found is feasible for the IP-Fixed formulation. In other words, the mappings generated by the IP-Fixed model and the heuristic have the same quality. In addition, the heuristic is much faster than the IP-Fixed formulation and can increase the flexibility by adding rows of pass-gates to solve instances that are infeasible with current row assignments, which cannot be achieved by the IP-Fixed model.

5.4 HOW MANY ROWS TO OPTIMIZE?

In the sliding partial MILP heuristic, a window of rows is selected to optimize. We tested different alternatives in order to decide how many rows to optimize in this window. The more rows are optimized, the longer the MILP takes. However, it may not be possible to solve the violations if too few rows are optimized. We consider 1 row, 2, 3, 4 or 5 rows, as well as some approaches that change the number of rows. We didn't go past 5 rows since it started to take long.

In optimizing a single ALU row, we adjust the columns of the operators in the selected row. Since all the variables are binary and locations of the operators in other rows are restricted, this formulation can be solved as an LP. There cannot be any basic feasible solutions that contain fractional values for this formulation. There are several options for

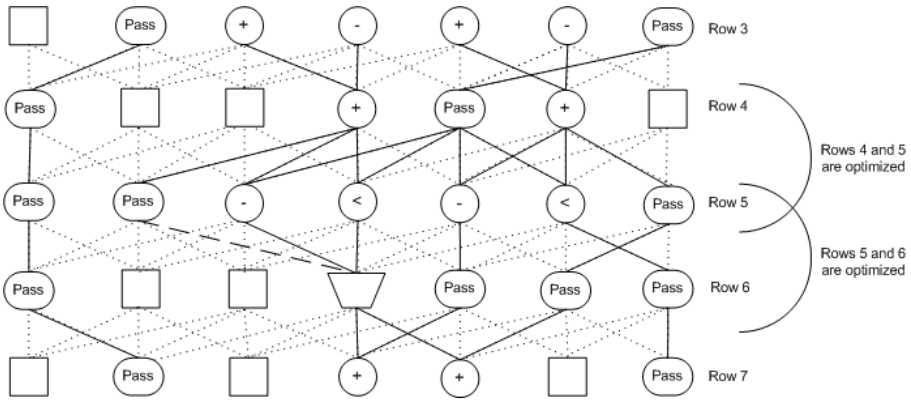


Figure 5.8: Optimization of 2 rows

which row to optimize: the starting row or the ending row of the violated edge. We can also try both iteratively: first optimize the top row, if this does not work, optimize the bottom row. All of these approaches have been implemented, however they are not able to solve most of the instances. This approach often starts to add rows of pass-gates in the same location again and again. Optimizing a single row does not provide enough flexibility to fix violations. Computational results are shown in Section 5.4.2.

In optimizing 2 rows, we consider two possibilities. We may adjust the columns of the operators in the rows where the violation occurs or we may adjust the columns in the previous windows—improved 2 rows. For example if there is a violation between rows 5 and 6, first rows 4 and 5 are optimized (Figure 5.8). If the edge cannot be fixed, rows 5 and 6 are optimized. This alternative is better, however neither of these alternatives can solve most of the instances.

We could choose 3 rows as the optimization window, see Figure 5.5 on page 46. Suppose there is a violated edge between rows 5 and 6. We restrict locations of the operators in rows 3 and 7, and optimize the locations of the operators in rows 4, 5 and 6. This approach can solve 9 of the 10 instances which are described in Section 5.4.2. It cannot solve one of the instances, since one of the operators has grandparents far from each other. It cannot even handle this instance by adding rows of pass-gates.

Having grandparents or great-grandparents far from each other causes problems. To address this problem we introduce graded objective coefficients. The farther the violation is, the more it will cost. Based on this idea, the objective coefficients are multiplied by the absolute distance difference between the column numbers of the edges' operators. Thus, operators with the same grandchildren are more likely to be placed close to each other. Even if the grandparents or great-grandparents are far from each other, violations can be fixed with the graded objective function by adding enough rows of pass-gates. After adding this feature, "Graded optimize 3" can solve all of the instances. However, it adds 3 rows of pass-gates for one of the instances, see Section 5.4.2.

Optimizing 3 rows is successful because the rows of pass-gates are utilized efficiently. When a row of pass-gates is added, locations of the operators in the preceding and succeeding rows can be adjusted. In contrast, when 2 rows are optimized, only the locations of the row of pass-gates and another row can be adjusted, which does not always help. In other words, sometimes the violation cannot be fixed and rows of pass-gates are constantly added between the same rows.

The 4 rows approach performs well. All of the instances are solved by adding at most one row of pass-gates. In addition, the run times which are shown in Section 5.4.2 are good.

Optimizing 5 rows can solve all of the instances by adding at most 2 rows. However the solution times are much longer than the "Optimize 4 rows" version and there is not much improvement in terms of the number of rows added.

5.4.1 Iterative Approaches

We also tried approaches that change. The number of rows optimized is increased if a violation cannot be fixed or pushed down. For instance, in "Iterative 1234" first one row is optimized, if a violation cannot be fixed, the locations of the operators in 2 rows are adjusted and we continue like this up to 4 rows being optimized. If optimizing 4 rows cannot solve the problem, a row of pass-gates is added. "Iterative 1234" and "Iterative 234" do not perform well. They add more rows than the "Optimize 4 rows" version. However "Iterative 34" is competitive with "Optimize 4 rows". Computational tests on the benchmark instances are summarized in Section 5.4.2.

5.4.2 Computational Tests

CPLEX 9.0 is used in the sliding partial MILP heuristic to solve the nearly feasible instances built from the IP-Fixed formulation and simulated annealing (SA) on a hyper-threaded Pentium 4 operating at 3.2 GHz with 4G of memory running Linux. These tests are for the cardinality 5 interconnect.

Tables 5.2 and 5.3 summarize the number of rows added and the run times of the sliding partial MILP heuristic for different alternatives. The first number in the name of the IP instances is the time we stop the IP-Fixed formulation and the second number is the number of violated edges. For SA instances, we only have the number of violations. In Table 5.2, the heuristic starts from a solution generated by the IP-Fixed formulation at the end of 10 minutes, which is found in Section 5.2 to be a good point to stop running the IP-Fixed model. In Table 5.3, the starting solution of the heuristic is the first integer solution found by the IP-Fixed for sobel and laplace since they are easy to solve, or the infeasible solution found by SA for encoder, idctcol and idctrow. SA can solve the other instances.

As can be seen from Tables 5.2 and 5.3, "Optimize 1 row" only solves 4 out of 10 instances and adds 22 rows of pass-gates in one case—decoder. "Optimize 2 rows" version solves 4 instances and cannot solve any of the instances generated by the SA. "Optimize 3 rows" solves 9 out of 10 instances. The longest one takes 37.78 seconds and it adds at most 5 rows. The first version that can solve all of the instances is "Graded optimize 3 rows". The longest

Table 5.2: Tests on MILP Instances. Times are in seconds.

IP Instances	Gsm_600_5		Encoder_600_5		Decoder_600_4		Idctcol_600_4		Idctrow_600_9	
Heuristics	Row	Time	Row	Time	Row	Time	Row	Time	Row	Time
Opt. 1 Row	-	-	3	0.13	22	1.42	1	0.14	4	0.15
Opt. 2 Rows	-	-	-	-	2	6.45	**0**	3.45	4	7.12
Opt. 3 Rows	**0**	15.29	**0**	3.78	-	-	**0**	8.39	1	9.22
Graded Opt. 3	**0**	4.60	**0**	3.37	**0**	25.78	**0**	6.84	1	5.94
Opt. 4 Rows	**0**	12.97	**0**	4.80	**0**	70.09	1	51.05	1	43.20
Graded Opt. 4	**0**	10.18	**0**	4.68	**0**	30.13	**0**	24.28	1	92.42
Opt. 5 Rows	**0**	21.35	**0**	19.96	**0**	80.99	**0**	412.41	**0**	209.16
Graded Opt. 5	**0**	11.13	**0**	10.08	**0**	17.96	**0**	71.98	**0**	117.49
Iterative 1234	**0**	17.95	**0**	7.30	4	44.64	**0**	5.93	1	67.49
Iterative 234	**0**	18.90	**0**	8.01	4	46.02	**0**	6.19	1	48.33
Iterative 34	**0**	30.48	**0**	6.56	2	45.68	**0**	15.63	1	79.98
Graded 34	**0**	7.31	**0**	5.66	**0**	10.83	**0**	11.25	1	51.62

one takes 31.60 seconds—encoder_SA. "Optimize 4 rows", "Graded optimize 4 rows" and "Optimize 5 rows" add at most one row of pass-gates. The longest instance takes 158.22 seconds for "Graded optimize 4 rows". However, the run times are longer for "Optimize 5 rows", for example idctrow_SA can be solved in 1725.46 seconds. Versions that cannot solve all of the instances should be ruled out. Among the rest there is a tradeoff between time and rows added.

In iterative approaches, the "decoder" instance causes problems for many approaches. Only "Graded 34" does not add a row of pass-gates, see Table 5.2. "Graded 34" is competitive with "Graded optimize 4 rows". The longest instance takes 89.30 seconds and "Graded 34" adds at most 2 rows of pass-gates.

Based on the computational tests so far, "Graded optimize 4 rows" and "Graded 34" are the best approaches in terms of balancing number of rows added and the run times.

5.4.2.1 Interconnects with Dedicated Pass-gates: We also use the sliding partial MILP heuristic for interconnects with dedicated pass-gates. Tables 5.4, 5.5 and 5.6 show the number of rows added and the run times for the 7 benchmarks for two different

Table 5.3: Tests on MILP & SA Instances. Times are in seconds.

IP & SA Ins.	Sobel_10_18		Laplace_5_14		Encoder_SA_7		Idctcol_SA_13		Idctrow_SA_7	
Heuristics	Row	Time	Row	Time	Row	Time	Row	Time	Row	Time
Opt. 1 Row	-	-	-	-	-	-	-	-	-	-
Opt. 2 Rows	5	2.13	-	-	-	-	-	-	-	-
Opt. 3 Rows	1	1.81	5	3.05	2	37.79	3	47.53	4	30.12
Graded Opt. 3	1	1.75	2	2.38	**0**	31.60	1	24.55	3	23.54
Opt. 4 Rows	**0**	5.76	1	3.54	1	391.33	**0**	218.10	**0**	75.73
Graded Opt. 4	**0**	10.18	**0**	4.68	**0**	161.43	**0**	121.06	1	158.22
Opt. 5 Rows	**0**	16.68	1	33.63	**0**	559.14	**0**	937.73	**0**	1725.46
Graded Opt. 5	**0**	27.87	**0**	78.02	**0**	516.25	**0**	1313.46	2	3367.97
Iterative 1234	**0**	8.58	**0**	6.30	**0**	33.29	1	138.66	**0**	68.73
Iterative 234	**0**	8.28	**0**	4.10	**0**	32.66	1	139.21	**0**	68.78
Iterative 34	**0**	8.86	**0**	4.10	**0**	86.18	**0**	254.9	**0**	43.71
Graded 34	**0**	6.53	**0**	3.46	**0**	44.49	1	49.00	2	89.30

interconnects—cardinality 8 and cardinality 5 interconnects with dedicated pass-gates, which are discussed in Section 3.5.1. The sliding partial MILP heuristic starts from a solution built by the IP-Fixed Formulation. The solutions are generated by the IP-Fixed Formulation at the end of 10 minutes or are the first integer solutions found by the IP-Fixed Formulation if it does not take 10 minutes to solve to optimality. We use the "Graded optimize 4 rows" approach in these tests.

As can be seen from Table 5.4, the heuristic is successful on every instance for cardinality 8 interconnect with dedicated pass-gates. It adds a row of pass-gates for only idctrow with the interconnect 8_8P whereas the IP-Fixed model cannot solve this instance in 10 hours. This is because the heuristic has an advantage of increased flexibility. The longest run time is 65.05 seconds, for a total of 665 seconds including the time spent on IP-Fixed.

Table 5.5 shows that 8 out of 21 instances cannot be solved by the sliding partial MILP heuristic for cardinality 5 interconnect with dedicated pass-gates. However, by adding rows of pass-gates the heuristic can solve four of the instances—encoder 55_8P, decoder 5_8P, laplace 55_8P and laplace 5_8P—which were proven infeasible for the IP-Fixed formulation. It adds at most 2 rows of pass-gates. When we consider the instances solved by the heuristic,

Table 5.4: Tests on cardinality 8 interconnect with dedicated pass-gates. Times are in seconds.

Graded Opt. 4 Rows	888_8P		88_8P		8_8P	
MILP Instances	Rows	Time	Rows	Time	Rows	Time
Gsm	0	30.38	0	19.13	0	32.59
Encoder	0	47.74	0	65.05	0	13.64
Decoder	0	2.20	0	13.85	0	3.37
Idctcol	0	28.02	0	4.04	0	42.09
Idctrow	0	23.63	0	28.51	1	54.72
Sobel	0	3.12	0	2.32	0	3.57
Laplace	0	1.36	0	4.13	0	5.27

the longest run time is 2130.5 seconds. Four of these instances cannot be mapped by the IP-Fixed formulation within an hour.

To better understand the heuristic's performance, we analyze the 8 instances which cannot be solved by the heuristic. The benchmarks idctcol and idctrow are infeasible because they have nodes which have 4 commutative children. In a situation like this, dedicated pass-gates cannot work with cardinality 5 interconnect. It is impossible to map sobel with the interconnect 5_8P since one of the nodes has 4 children. Coder cannot be mapped with the interconnect 5_8P since it has a node which has 3 "mux" children. These infeasible structures can be seen in Figure A8 on page 125. To be able to solve idctcol, idctrow and sobel we revise the operator assignments to rows such that a node can have at most 3 children. After this revision the heuristic can solve the 7 instances. The results can be seen in Table 5.6. This shows us that the value of maximum fan-out (number of children) should be chosen carefully, otherwise some interconnects cannot be used with every instance.

The interconnect 5_8P is very restrictive since 50% of the ALUs are used as dedicated pass-gates and the ALUs use cardinality 5 interconnect. The heuristic adds 5 rows for idctcol

Table 5.5: Tests on cardinality 5 interconnect with dedicated pass-gates. Times are in seconds.

Graded Opt. 4 Rows	555_8P		55_8P		5_8P	
MILP Instances	Rows	Time	Rows	Time	Rows	Time
Gsm	0	0.68	0	60.45	1	39.90
Encoder	0	19.06	**2**	321.84	-	
Decoder	0	4.70	0	41.57	**2**	2130.15
Idctcol	-		-		-	
Idctrow	-		-		-	
Sobel	0	0.70	0	7.45	-	
Laplace	0	0.87	**1**	203.22	**2**	116.01

and 6 for idctrow to generate a mapping. However, the sliding partial MILP heuristic solves the other interconnects—555_8P, 55_8P —by adding at most 2 rows and the longest one takes 376.25 seconds.

5.4.2.2 Different Starting Solutions: We also test the sliding partial MILP heuristic with different starting solutions. These solutions are generated by the IP-Fixed formulation at different times and have diverse numbers of violated edges. The results of tests for "Optimize 4 rows" and "Graded optimize 4 row" can be seen in Table 5.7. "Optimize 4 rows" and "Graded optimize 4 row" are used since they gave the best results in the computational tests of Section 5.4.2. The first number in the name of the instances is the run time at which the integer solution is found and the second number is the number of violated edges. For example integer solution "encoder_160_162" is found at 160 seconds and has 162 violations. The instances laplace and gsm have only one integer solution in 600 seconds and tests on these instances were already shown in Tables 5.2 and 5.3.

Table 5.6: Revised tests on cardinality 5 interconnect with dedicated pass-gates. Times are in seconds.

Graded Opt. 4 Rows	555_8P		55_8P		5_8P	
MILP Instances	Rows	Time	Rows	Time	Rows	Time
Gsm	0	0.68	0	60.45	1	39.90
Encoder	0	19.06	**2**	321.84	-	
Decoder	0	4.70	0	41.57	**2**	2130.15
Idctcol	**2**	376.25	**2**	336.44	**5**	485.38
Idctrow	**1**	35.18	**1**	190.15	**6**	2221.16
Sobel	0	0.70	0	7.45	0	7.10
Laplace	0	0.87	**1**	203.22	**2**	116.01

"Graded optimize 4 rows" adds at most 3 rows of pass-gates and the longest instance takes 449.04 seconds, see Table 5.7. "Optimize 4 rows" adds at most 5 rows and the longest instance takes 175.97 seconds. The heuristic can even fix 162 violated edges—encoder_160_162—while only adding one row of pass-gates. This shows the strength of the heuristic. It is not dependent on starting from a near-optimal solution. Starting from an arbitrary solution will be discussed in the next section.

5.5 STARTING FROM AN ARBITRARY SOLUTION

The sliding heuristic was initially intended to be used as a post-processor for nearly feasible solutions, but the results of the preceding section show that it can handle many violated edges. In this section, we test it on instances starting from an arbitrary infeasible solution. Thus, the heuristic does not need any other algorithm to generate a valid mapping. Instead of using the nearly feasible MILP or SA solutions, the heuristic starts with an arbitrary

Table 5.7: Tests on other MILP Instances.

Times in seconds	Optimize 4 Rows		Graded Optimize 4 Rows	
MILP Instances	Rows Added	Run Time	Rows Added	Run Time
encoder_160_162	1	122.43	3	449.04
encoder_317_61	2	175.97	1	359.41
Encoder_600_5	0	4.80	0	4.68
decoder_163_29	0	57.63	0	92.90
decoder_385_6	0	6.18	0	12.43
Decoder_600_4	0	70.09	0	30.13
idctcol_127_42	1	70.10	0	140.70
idctcol_360_26	1	70.50	0	134.79
idctcol_578_6	1	15.40	1	62.67
Idctcol_600_4	1	51.05	0	24.28
idctcol_719_3	0	46.97	0	73.75
idctrow_80_99	5	105.53	2	316.46
idctrow_347_77	2	102.05	2	344.03
Idctrow_600_9	1	43.20	1	92.42
sobel_3_19	3	27.04	0	14.35
sobel_10_18	0	10.79	0	10.18

placement where operations are placed in the earliest row possible and the operations are left justified. For a specific row the first operator is assigned to column 1, the second operator is assigned to column 2 and the remaining operators are located similarly. These assignments will have many violations.

In the sliding partial MILP heuristic, it is sometimes necessary to add rows of pass-gates to fix violations. When this option is removed, we have a heuristic which runs partial MILPs and tries to minimize the number of violated edges. The number of MILPs depends on the number of rows. In this heuristic, we do not need to fix every violation, we just want to get a good nearly feasible solution without adding rows of pass-gates. We will use this heuristic on the arbitrary solutions to have better starting points for the sliding partial MILP heuristic. The algorithm for the heuristic can be seen below:

Heuristic n rows:

for i=1 to r-n+1 **do**

Solve a partial MILP to minimize the number of violation(s) in rows i, i+1,..., i+n-1.

end for

We try three alternatives: heuristic 1 row, heuristic 2 rows and heuristic 3 rows. For example in "Heuristic 2 rows", first the locations of the operators in row 3 are restricted and the columns of the operators in rows 1 and 2 are adjusted, then the rows 1 and 4 are restricted, and rows 2 and 3 are optimized. We continue similarly until the last 2 rows are optimized. "Heuristic 1 row" and "Heuristic 3 rows" are similar, only the number of rows optimized in each window changes. We show the results of the computational tests in the next section.

5.5.1 Computational Tests

In this section, we present the results of tests on the sliding partial MILP heuristic starting from arbitrary solutions and solutions generated by the heuristic which does not add any rows of pass-gates.

Table 5.8 summarizes the number of rows added and the run times for the heuristics "Optimize 1 row", "Optimize 2 rows" and "Optimize 3 rows" starting from an arbitrary

Table 5.8: Tests on arbitrary instances.

Times in seconds	Optimize 1 row		Optimize 2 rows		Optimize 3 rows	
Arbitrary Ins.	Rows	Time	Rows	Time	Rows	Time
Gsm	> 20	-	> 20	-	1	30.89
Encoder	> 20	-	> 20	-	3	40.47
Decoder	> 20	-	> 20	-	2	31.24
Idctcol	> 20	-	> 20	-	7	102.28
Idctrow	> 20	-	> 20	-	2	21.99
Sobel	4	0.06	3	4.25	0	4.80
Laplace	3	0.04	3	3.48	1	95.07

solution. "Optimize 1 row" and "Optimize 2 rows" only solve 2 small benchmarks—sobel and laplace. Other instances cannot be solved by adding 20 rows of pass-gates. However, "Optimize 3 rows" solves all of the instances. The longest run time is 102.28 seconds and it adds at most 7 rows.

Table 5.9 summarizes the number of rows added and the run times for the "Optimize 4 rows" and "Graded optimize 4 rows". "Graded optimize 4 rows" solutions are always as good as the "Optimize 4 rows" solutions in terms of fabric size. In 3 out of 7 instances—encoder, idctcol, idctrow—it adds fewer rows. There are not significant differences in the run times. Even though we start from an arbitrary solution the run times are not that long: the longest one takes around 10 minutes.

The tests on "Graded 34" and "Graded optimize 5" heuristics can be seen in Table 5.10. When we compare "Graded optimize 4" and "Graded 34", "graded optimize 4" is better for gsm and "Graded 34" is better for idctcol in terms of number of rows added. The run times of both versions are close. On the other hand, "Graded optimize 5" adds fewer rows than the other heuristics. In fact, it adds at most 2 rows of pass-gates. However the run times are longer—especially for idctcol and idctrow—as expected and it is not a practical option.

Table 5.9: "Optimize 4 Rows" tests on arbitrary instances.

Times in seconds	Optimize 4 rows		Graded Optimize 4 rows	
Arbitrary Ins.	Rows Added	Run Time	Rows Added	Run Time
Gsm	0	42.60	0	79.80
Encoder	3	151.60	2	142.68
Decoder	0	150.15	0	32.22
Idctcol	6	673.82	5	635.38
Idctrow	2	342.58	1	373.11
Sobel	0	2.61	0	3.86
Laplace	0	11.68	0	87.27

Table 5.10: "Iterative 34" and "Optimize 5 Rows" tests on arbitrary instances.

Times in seconds	Graded 34		Graded Optimize 5	
Arbitrary Instances	Rows Added	Run Time	Rows Added	Run Time
Gsm	1	40.46	0	258.47
Encoder	2	93.40	1	317.66
Decoder	0	43.39	0	33.82
Idctcol	3	242.29	2	1597.75
Idctrow	1	48.27	1	5720.9
Sobel	0	6.73	0	14.76
Laplace	0	89.68	0	159.05

Table 5.11: Heuristic run times.

Time (seconds.)	Heuristic 1 row	Heuristic 2 rows	Heuristic 3 rows
Gsm	0.05	7.36	35.71
Encoder	0.05	9.54	42.08
Decoder	0.03	4.19	51.27
Idctcol	0.05	9.29	106.17
Idctrow	0.03	5.38	42.30
Sobel	0.01	1.45	5.71
Laplace	0.02	1.23	10.04

Based on all of the computational tests, graded objective coefficients help to find mappings with fewer rows and faster run times. In addition, "Optimize 4" and "Iterative 34" are the best approaches overall.

Instead of starting from an arbitrary solution in the sliding partial MILP heuristic, we can run the "Heuristic 1 row", "Heuristic 2 rows" or "Heuristic 3 rows" and generate a nearly feasible solution. Table 5.11 gives the run times of these heuristics starting from arbitrary solutions. Run times are less than 1 second for "Heuristic 1 row", less than 10 seconds for "Heuristic 2 rows" and less than 2 minutes for "Heuristic 3 rows". The resulting solutions are used as the starting points for the sliding partial MILP heuristic. Table 5.12 summarizes the number of rows added and the run times of the "Optimize 4 rows" version starting from these solutions. Starting the sliding partial MILP heuristic from "Heuristic 1 row" or "Heuristic 2 rows" solutions is not much better than starting from an arbitrary solution. However, "Heuristic 3 rows" solutions perform better. The sliding partial MILP heuristic starting from "Heuristic 2 rows" or "Heuristic 3 rows" adds fewer rows and solution times are shorter than starting from an arbitrary solution. Total run times for starting from the heuristic solutions are less than 4 minutes whereas run times for starting from arbitrary solutions are less than 11 minutes.

Table 5.12: Tests on optimized instances. Times are in seconds.

Optimize 4 Rows	Heuristic 1 row		Heuristic 2 rows		Heuristic 3 rows	
Instances	Rows	Time	Rows	Time	Rows	Time
Gsm	0	4.38	1	4.24	0	2.60
Encoder	2	78.33	1	104.04	0	81.10
Decoder	0	47.92	1	107.87	0	38.82
Idctcol	3	195.01	2	140.17	1	90.22
Idctrow	1	111.83	1	151.23	1	25.20
Sobel	0	0.62	0	4.86	0	0.32
Laplace	0	13.33	0	35.62	0	0.68

We used different starting points for the sliding partial MILP heuristic: IP-Fixed, SA, arbitrary, heuristic. Starting from IP-Fixed or SA solutions adds the fewest rows and total run times (IP-Fixed/SA+sliding partial MILP heuristic) are less than 12 minutes. When the sliding partial MILP heuristic starts from an arbitrary solution, it adds more rows and solution times are less than 11 minutes. Starting from heuristic solutions add competitive numbers of rows and total run times are less than 4 minutes. So, if generating mappings in less than 5 minutes is essential, the best option is running the "Heuristic 3 rows" to generate a starting point and then using the "Graded optimize 4 rows" sliding partial MILP heuristic to generate a valid mapping. If 5-12 minutes run times are acceptable, the best option is starting the sliding partial MILP heuristic from IP-Fixed or SA solution.

5.6 COMPARISON WITH THE GREEDY HEURISTIC

In this section, the sliding partial MILP heuristic is compared with the greedy heuristic which is discussed in Section 3.6.2. Cardinality 5 interconnect and interconnects with dedicated pass-gates are considered.

We are concerned with three aspects of the solutions:

- Size: How many rows are added?
- Path length: How much is the length of the paths increased?
- Time: How much computation time is necessary to identify mappings?

The path length is the total number of active edges from the inputs to the outputs. The fabric size and the path length are essential because they have significant impact on energy consumption. On the other hand, run time is important since we do not want to spend too much time mapping the operators and connections onto the fabric. The first two aspects are similar especially for the sliding partial MILP heuristic. When a row of pass-gates is added, the path length increases by the number of outputs affected by this change. However, it is different in the greedy heuristic since it does not add a complete row of pass-gates.

Tables 5.13 and 5.14 give the number of rows needed for cardinality 5 interconnect for the greedy heuristic and the sliding partial MILP heuristic starting from different solutions. In terms of fabric size, the sliding partial MILP heuristic is strictly better than the greedy heuristic for all of the instances no matter which starting point is used. Even if the sliding partial MILP heuristic starts from an arbitrary assignment, the fabric size is always smaller. "IP-Fixed+Sliding partial MILP heuristic" and "SA+Sliding partial MILP heuristic" give the best results.

Tables 5.15, 5.16, 5.17 and 5.18 show the path length increases for cardinality 5 interconnect and interconnects with dedicated pass-gates. The sliding partial MILP heuristic is better than the greedy heuristic for cardinality 5 interconnect and for cardinality 8 interconnect with dedicated pass-gates. For cardinality 8 interconnect with dedicated pass-gates, the path length increases by 8 for only idctrow 8_8P. For cardinality 5 interconnect with dedicated pass-gates, the path length increase of idctrow 5_8P is higher than the increase

Table 5.13: Fabric size comparison with the greedy heuristic for cardinality 5 interconnect.

# of Rows **Cardinality 5**	Greedy Heuristic	IP-Fixed+ Sliding MILP Heuristic	SA+Sliding MILP Heuristic
Gsm	19	**18**	**18**
Encoder	21	**16**	**16**
Decoder	14	**13**	**13**
Idctcol	20	**13**	**13**
Idctrow	14	**11**	**11**
Sobel	10	**9**	**9**
Laplace	10	**8**	**8**

Table 5.14: Fabric size comparison with the greedy heuristic for cardinality 5 interconnect.

# of Rows **Cardinality 5**	Greedy Heuristic	Arbitrary+Sliding MILP Heuristic	Heuristic+Sliding MILP Heuristic
Gsm	19	**18**	**18**
Encoder	21	19	**16**
Decoder	14	**13**	**13**
Idctcol	20	18	14
Idctrow	14	12	**11**
Sobel	10	**9**	**9**
Laplace	10	**8**	**8**

Table 5.15: The difference in total path length for cardinality 5 interconnect.

Path Length Cardinality 5	Greedy Heuristic	IP-Fixed+ Sliding MILP Heuristic	SA+Sliding MILP Heuristic
Gsm	2	0	0
Encoder	11	0	0
Decoder	1	0	0
Idctcol	42	0	0
Idctrow	26	8	8
Sobel	1	0	0
Laplace	2	0	0

Table 5.16: The difference in total path length for cardinality 5 interconnect.

Path Length Cardinality 5	Greedy Heuristic	Arbitrary+Sliding MILP Heuristic	Heuristic+Sliding MILP Heuristic
Gsm	2	0	0
Encoder	11	6	0
Decoder	1	0	0
Idctcol	42	40	8
Idctrow	26	8	8
Sobel	1	0	0
Laplace	2	0	0

Table 5.17: The difference in total path length for cardinality 8 interconnect with dedicated pass-gates.

Increase	888_8P		88_8P		8_8P	
Instances	Greedy Heuristic	IP+ Sliding IP Heuristic	Greedy Heuristic	IP+Sliding IP Heuristic	Greedy Heuristic	IP+Sliding IP Heuristic
Gsm	0	0	0	0	0	0
Encoder	0	0	0	0	2	0
Decoder	0	0	0	0	0	0
Idctcol	1	0	3	0	5	0
Idctrow	1	0	0	0	11	8
Sobel	0	0	0	0	0	0
Laplace	0	0	1	0	2	0

when the greedy heuristic is used and there is no solution for encoder 5_8P. The path length increase is lower for all of the other instances, see Table 5.18.

Tables 5.19 and 5.20 give the run times for cardinality 5 interconnect. The run time $600+10$ for the gsm means 600 seconds is spent to find a nearly feasible integer solution and the sliding partial MILP heuristic takes 10 seconds to generate a valid mapping. The greedy heuristic algorithm has the best execution times with none over 10 seconds. The longest total time of using the sliding partial MILP heuristic on MILP instances is 12 minutes, on SA instances is 5 minutes, on arbitrary instances is 11 minutes and on "Heuristic 3 rows" instances is 4 minutes.

Table 5.18: The difference in total path length for cardinality 5 interconnect with dedicated pass-gates.

Increase	555_8P		55_8P		5_8P	
Instances	Greedy Heuristic	IP+ Sliding IP Heuristic	Greedy Heuristic	IP+Sliding IP Heuristic	Greedy Heuristic	IP+Sliding IP Heuristic
Gsm	0	0	2	0	4	2
Encoder	6	0	11	4	10	-
Decoder	2	0	3	0	4	4
Idctcol	44	16	37	16	45	40
Idctrow	35	8	33	8	26	48
Sobel	1	0	0	0	3	0
Laplace	2	0	2	1	4	2

Table 5.19: Run time comparison with the greedy heuristic for cardinality 5 interconnect.

Run Time Cardinality 5	Greedy Heuristic	IP-Fixed+ Sliding MILP Heuristic	SA+Sliding MILP Heuristic
Gsm	3	600+10	125
Encoder	10	600+4	160+161
Decoder	4	600+30	82
Idctcol	8	600+24	165+121
Idctrow	6	600+92	120+158
Sobel	1	10+10	25
Laplace	1	5+4	27

Table 5.20: Run time comparison with the greedy heuristic for cardinality 5 interconnect.

Run Time Cardinality 5	Greedy Heuristic	Arbitrary+Sliding MILP Heuristic	Heuristic+Sliding MILP Heuristic
Gsm	3	79	35+2
Encoder	10	142	42+81
Decoder	4	32	51+38
Idctcol	8	635	106+90
Idctrow	6	373	42+25
Sobel	1	3	5+1
Laplace	1	87	10+1

Based on the computational tests the sliding partial MILP heuristic performs better than the greedy heuristic in terms of fabric size and path length. However, the run times for the greedy heuristic are shorter. If generating mappings in seconds is essential, the greedy heuristic should be used. If energy consumption is critical and 0-12 minutes run times are acceptable, the sliding partial MILP heuristic should be used. In general, energy consumption is more important.

5.7 CONCLUSION

In this chapter, the sliding partial MILP heuristic was presented. The heuristic is useful in several situations. When the MILP formulation takes too long to solve, we can stop at a nearly feasible solution and apply the heuristic. The MILP instances for IP-Fixed are sometimes infeasible with current row assignments, the heuristic can overcome this issue by adding rows of pass-gates. Other researchers in our group use simulated annealing to generate mappings, but SA does not generate valid mappings for some of the instances. Using the heuristic as a post-processor makes their approach usable.

The sliding partial MILP heuristic generates mappings by using small, fast MILPs. It gives an optimal solution like the IP-Fixed formulation if it does not add any rows of pass-gates. As a result, we show that sometimes large MILPs that take too long can be partitioned into smaller MILPs and can be solved faster.

Among the different alternatives tried, "Graded optimize 4" and "Graded 34" are the best approaches for the sliding partial MILP heuristic. They add fewer rows—most of the time less than 3 rows—and the run times are good—less than 12 minutes. The heuristic behaves well even if the starting solution is arbitrary.

The sliding partial MILP heuristic adds fewer rows than the greedy heuristic. In addition, the total path length is shorter when the sliding partial MILP heuristic is used. However, the greedy heuristic is faster than the sliding partial MILP heuristic.

In our computational tests, we have considered instances with up to 20 rows and 20 columns. Future work will focus on larger instances.

6.0 AUTOMATED TUNING OF OPTIMIZATION SOFTWARE PARAMETERS

As discussed in Chapter 5, we expect the CAD community would not accept the run times for the IP-Fixed formulation. Twelve out of 91 mapping instances cannot be solved within 10 hours. One of the ways to shorten the execution times is to tune the parameters of the MILP solvers. In this chapter, we present a method to tune software parameters using ideas from software testing and machine learning. We will use this method to identify good parameter values for CPLEX, CBC and GLPK. The method is based on the key observation that for many classes of instances, the software shows improved performance if a few critical parameters have "good" values, although which parameters are critical depends on the class of instances. Our method attempts to find good parameter values using a relatively small number of optimization trials.

The full method is the result of a collaboration with Brady Hunsaker, Paul Brooks and Abhijit Gosavi. This chapter presents the contributions of the author which include a key observation about when our approach is reasonable, implementation of pairwise coverage algorithm and different metrics, discovering good configurations of our method itself and computational tests on various instances.

The remainder of this chapter is organized as follows. Section 6.1 considers the motivation behind the automated tuning of software parameters. We describe the problem more formally in Section 6.2. Section 6.3 provides a literature review. We discuss the key observation in Section 6.4. The general framework of the algorithm is explained in Section 6.5. Computational tests and results on various instances are presented in Sections 6.6 and 6.7. Finally, concluding remarks are provided in Section 6.8.

6.1 MOTIVATION

MILP solvers depend on parameters that must be set before the program is executed. The performance of the software often depends significantly on the values chosen for these parameters. We consider three MILP solvers: GLPK 4.11, CBC 1.01 and CPLEX 9.0. GLPK has 11 algorithmic parameters. CBC has more than 20 parameters, including a growing number of cut parameters. There are dozens of algorithmic parameters in CPLEX 9.0, including 9 kinds of cuts alone.

The number of possible parameter combinations for these solvers is very high. When we consider a subset of just 6 important parameters for CPLEX—4 parameters with 3 values each and 2 parameters with 4 values each—there are 1296 possible combinations. Considering a subset of 10 important parameters yields over 100000 possible combinations. MILP solvers provide this great flexibility because parameter settings that are good for one type of instance may be bad for another type of instance. This flexibility is important because it is impossible to find settings that perform well for every class of instances. Our experience is that for any particular class of instances, there are generally settings better than the default setting.

The flexibility provided by having many parameters poses the significant challenge of identifying parameter values that are good for the instances that a user wants to solve. We present a method to help users address this challenge. Our method tries a number of settings on representative instances and reports the best setting observed. We implemented this method in a tool which we call Selection Tool for Optimization Parameters (STOP).

Because it tries many settings, our method requires a significant investment of computer time, though it requires very little of the user's time. For example, with 64 settings, a single instance, and a time limit of one hour, it could take up to 64 hours of CPU time. There are at least four scenarios when this initial investment of computer time makes sense:

- The user may want to run hundreds of similar instances, so the initial investment of computer time can be recovered from later time savings.
- The user may want to solve instances in "real time", when time is critical. In this situation, it may be worth investing lots of "off-line" time with similar instances to better utilize the user's "on-line" time.

- The user may need to solve a difficult instance, which possibly will require a lot of computational time, but also has a smaller instance of a similar type. Therefore, the software parameter can be tuned utilizing the smaller instance and this initial investment of computational time can be recovered from solving the more difficult instance faster.
- The user may be comparing alternative algorithmic ideas. In this case, he or she may want a fair comparison where all the approaches have good parameter settings. In this case, the time spent finding good parameter values is not critical.

The first and third scenarios above are valid for the mapping problem. There are different interconnects—cardinalities 5, 8, dedicated pass-gates, heterogeneous ALUs—and several benchmarks, therefore we will repeatedly solve similar instances. In addition, there are small and large benchmark instances. STOP can be tuned with smaller instances and resulting setting can be used for the larger instances. It is also important to generate mappings faster for the CAD community.

6.2 PROBLEM STATEMENT

We first describe the terminology we use: A software program has several *parameters* controlling its execution. Each parameter is assigned a particular *parameter value*. A set of parameter values—one for each parameter—is a *setting*. Informally, our goal is to find a good setting.

In order to use our method, the user specifies a set of instances to test. It is best to consider several instances with different sizes in order to represent the instance type better and avoid over-fitting the parameters. The user also states the number of settings that may be tried along with an optional solver time limit. These values depend on how much computer time the user is willing to use. The greater the number of settings tried, the more likely it is that better settings will be found. The optional solver time limit is the maximum time that each instance will be run with each setting; this limit avoids wasting a great deal of time on a setting that is clearly bad.

The parameters, parameter values to consider, and metric for comparison are specified in a solver interface file. The user may use an existing interface or may modify an interface to consider parameters and parameter values he or she believes to be important. It is possible to use our method with different metrics, such as solution time or solution quality. The metric time-to-optimality is the sum of the solution times of the test instances.

Our method assumes that each parameter has a small discrete set of values. This is the case for the MILP solvers we study. Another assumption is that good settings for the test instances will be good settings for similar types of instances. In Section 6.6.3, we present computational tests supporting this assumption.

6.3 LITERATURE REVIEW

Previous work on automated parameter tuning has often focused on parameters for particular algorithms using methods that do not generalize to the MILP branch-and-cut algorithms that motivate us. Examples of such work include [1, 18, 23]. Of these, the most general is [23], which uses an idea similar to local search, testing settings in a neighborhood of the current setting. Audet and Orban [4] provide a more general framework for parameter tuning, using mesh adaptive direct search to search through the space of settings. Both of these methods require a notion of parameter values that are "close" to one another, such as numeric parameter values, and the tuning is done by moving through the space of settings. In contrast, the MILP solvers we consider have many parameters that take discrete values with no clear order or relationship, so that moving through the space of settings does not seem natural. Our approach is instead based on an intelligent sampling of settings throughout the space.

Table 6.1: Key CPLEX parameters for mapping instances.

	Settings					
CPLEX (mapping)	1	2	3	4	5	Default
MIP emphasis	**1**	**1**	**1**	**1**	**0**	**0**
Node selection	**0**	**0**	**0**	**2**	**2**	**0**
Branching var. sel.	0	0	1	2	1	0
Dive type	1	0	0	3	3	0
Fractional cuts	**0**	**0**	**0**	**0**	**0**	**1**
MIR cuts	0	1	1	0	2	1
Time (s)	63.41	63.42	63.72	72.21	96.49	98.84

6.4 KEY OBSERVATION

In preliminary tests, we found that there are typically a small number of parameters that significantly improve the solution time of instances. However, which parameters are important depends on the instance class. If we can find good values for these few parameters, the solution time is relatively short. Our goal is to find a setting for which the critical parameter values are good, even though we may not know which parameters are critical.

To demonstrate this idea, Table 6.1 shows results for a set of mapping instances on CPLEX with six different parameter settings based on six parameters. The six parameters considered are MIP emphasis, node selection, branching variable selection, dive type, fractional cut generation (Gomory fractional cuts), and MIR cut generation. The exact purpose of these parameters is not important to demonstrate the key observation of our approach. More information about them can be found in the CPLEX user documentation.

As can be seen from the table, when we have the correct combination of the parameter values 1 (feasibility) for MIP emphasis, 0 (best-bound) for Node Selection and 0 (off) for Fractional Cuts, the solution time is low. When any of these three is not at these values,

Table 6.2: Key CBC parameters for p instances.

	Settings					
CBC (p)	1	2	3	4	5	Default
Strong Branching	0	2	1	2	1	1
Cost strategy	**1**	**1**	**1**	**1**	**1**	**3**
Gomory cuts	**0**	**0**	**0**	**1**	**3**	**3**
MIR cuts	0	3	2	3	2	3
Probing	3	2	0	0	3	3
Clique cuts	3	1	3	1	3	3
Feasibility pump	**0**	**0**	**0**	**0**	**1**	**1**
Time (s)	5.78	5.84	6.43	14.02	17.56	22.46

the solution time is longer. When two of them or all three of them are not at these values, the solution time is even longer.

Similarly, Table 6.2 shows results for a different set of instances on CBC. The seven parameters we explored are strong branching, cost strategy, Gomory cut generation, MIR cut generation, probing, clique cut generation, and feasibility pump. The instances are a subset of the 'p' instances from MIPLIB 3.0. In this case, cost strategy, Gomory cut generation, and feasibility pump are the three critical parameters that affect the solution time. The solution time is low for settings 1, 2 and 3 in Table 6.2 since the key parameters are at these good values. On the other hand, Gomory cut generation for setting 4, Gomory cut generation and feasibility pump for setting 5, and all three of the critical parameters for the default setting are not at these good values, which cause longer solution times.

If the critical parameters (for an instance class) have good values, then the non-critical parameters may take any value and the resulting setting will be "good". If the number of critical parameters is small, then this means that there are many "good" settings. Our goal is to find one of these settings without too many trials. Note, however, that we do not

necessarily expect to determine which parameters are critical for the instance class, since that is not our primary goal.

6.5 ALGORITHM

This section presents a high-level description of the algorithm [6]. This section is for completeness, it includes the work of other contributors.

Below is the general framework of our method:

1. Select initial settings to try.
2. Run the solver on those settings.
3. Use some form of machine learning to find additional settings.
4. Run the solver on additional settings.
5. Output the best setting observed.

The first step of the framework is the selection of initial settings. For this, we consider three options: random, greedy heuristic and pairwise coverage. The next step is to run the solver on those initial settings and record the solution times (or other metrics). Steps 3 and 4 are optional. In step 3, we use machine learning to guide our choice of additional settings based on the initial data. We consider two options for machine learning: regression trees and artificial neural networks. At the end, we report the best setting observed.

6.5.1 Selection of Settings

Based on the key observation from Section 6.4, we would like to have a good "spread" of settings. A motivating idea for selecting settings comes from orthogonal arrays. An orthogonal array of strength t is an $N \times m$ matrix in which the N rows represent settings with m parameters (one per column) such that for any t columns, all possible combinations of parameter values appear equally often in the matrix [43]. In our case this would mean that every combination of t parameter values appear equally often. Unfortunately, orthogonal

arrays do not exist for every combination of parameter and parameter value cardinalities and have not been well-studied in cases where parameters have different numbers of values.

We have tried three methods for selecting settings: pairwise coverage, a greedy heuristic, and random. Pairwise coverage is related to orthogonal arrays. It is widely used in software testing for identifying programming errors that are revealed when two parameter values interact. In our implementation, the goal is to have all pairs of parameter values appear at least twice in settings. An algorithm is used to identify settings that achieve this. If the number of settings needed is higher than the number the user has allowed, then we consider the easier requirement that all pairs appear at least once. We implemented an algorithm from [12] to generate the pairwise coverage test sets.

The second alternative is a greedy heuristic. The first setting is selected randomly. For each additional setting, all possible settings are compared with the already selected settings. For each possible setting, we consider the number of parameter values in common with each of the already-selected settings. We choose the new setting that minimizes the maximum number in common. Ties are broken based on the sum over the selected settings of the number of parameter values in common.

The final option for selecting settings is random selection. For each parameter, we select a value uniformly at random. This method is easy to implement, but it may miss some parameter interactions.

6.5.2 Machine Learning

Machine learning is an optional part of the algorithm. The goal of machine learning for our method is to extract useful information from the initial settings to guide our choice of additional settings by building good probabilistic models. We try two machine learning alternatives: regression trees and artificial neural networks.

Regression trees are used as a non-parametric method (i.e., no assumptions are made about the distribution of the data) for regression. They are developed as part of classification and regression trees (CART) in [10]. As in linear regression, regression trees are used to investigate the relationship between a continuous dependent variable and independent vari-

ables. The independent variables can be continuous or discrete. More information about this alternative can be found in [6]. Regression trees for our implementation are generated using the *rpart()* function in the *rpart* package in the R environment for statistical computing [36].

Artificial neural networks use a simple model of multi-layered neurons as a basis for predicting the value of a function. In our case, the function of interest takes the solver parameter values as inputs and gives the total run time as output. The neural network is trained using a number of observed function values (total running times) and then provides predicted running times for other inputs. For more details on artificial neural networks, see [6, 33, 38]. In our implementation, we use the free and open-source Fast Artificial Neural Network (FANN) library [35].

6.6 COMPUTATIONAL TESTS

We have implemented our approach in a tool we call STOP. We performed tests with three MILP solvers, using a subset of parameters available in each on a hyper-threaded Pentium 4 operating at 3.2 GHz with 4G of memory running Linux. For some parameters, our interfaces did not consider every possible parameter value. This is because the performance of the algorithm seems to be more sensitive to the numbers of parameter values than to the number of parameters. We do not explore this issue here, however, since the purpose is to demonstrate the usefulness of the algorithm and not to explore in detail the relationship of its effectiveness to the number of parameters and parameter values.

The exact parameters and parameter values considered in computational tests are given below.

- CPLEX 9.0 with six parameters
  - mip emphasis (CPX_PARAM_MIPEMPHASIS): automatic, feasibility, optimality, moving best bound
  - node selection (CPX_PARAM_NODESEL): best-bound, best-estimate, alternative best-estimate

  - branching variable (CPX_PARAM_VARSEL): automatic, pseudo costs, strong branching
  - dive type (CPX_PARAM_DIVETYPE): automatic, traditional dive, probing dive, guided dive
  - fractional cuts (CPX_PARAM_FRACCUTS): off, automatic, on
  - MIR cuts (CPX_PARAM_MIRCUTS): off, automatic, on
- CPLEX 9.0 with ten parameters
  - the six listed above
  - disjunctive cuts (CPX_PARAM_DISJCUTS): off, automatic, on
  - clique cuts (CPX_PARAM_CLIQUES): off, automatic, on
  - node presolve selector (CPX_PARAM_PRESLVND): off, automatic, force
  - probing (CPX_PARAM_PROBE): off, automatic, on
- CBC 1.01 with seven parameters
  - strong branching (strong): 0, 5, 9
  - cost strategy (cost): off, priorities
  - gomory cuts (gomory): off, on, root, ifmove
  - MIR cuts (mixed): off, on, root, ifmove
  - probing (probing): off, on, root, ifmove
  - clique cuts (clique): off, on, root, ifmove
  - feasibility (feas): off, on
- GLPK 4.11 with six parameters
  - scaling (LPX_K_SCALE): none, equilibration, geometric mean, geometric then equilibration
  - pricing (LPX_K_PRICE): largest coefficient, steepest edge
  - branching (LPX_K_BRANCH): first variable, last variable, drieback-tomlin, most fractional
  - backtrack (LPX_K_BTRACK): depth first, breadth first, best projection, best bound
  - lp presolve (LPX_K_PRESOL): off, on
  - method: old b&b (lpx_integer), new b&b (lpx_intopt), new b&b with cuts (lpx_intopt and LPX_K_USECUTS)

We used 4 sets of instances: p and misc instances from MIPLIB 3.0 [9], aircraft landing instances from OR-Library [8], and mapping instances which are described in Chapter 4.

- p instances: p0033, p0201, p0282
- misc instances: misc03, misc06, misc07
- aircraft landing instances:
  - easy: airland2-R2, airland4-R4, airland7-R1
  - medium: airland4-R2, airland5-R1, airland5-R2
- mapping instances: sobel, laplace, gsm

The primary purposes of this section are to demonstrate that our methods can find settings better than the default settings for a variety of solvers and instance classes and to demonstrate that the resulting settings are also good on other instances from the same instance classes. To this end, Section 6.6.2 compares the results from STOP with those of the default settings on the tested instances. Section 6.6.3 looks at the question of whether good settings for the test instances are good for related instances. Prior to those results, we briefly consider one internal issue. Section 6.6.1 compares some of the many options within STOP.

6.6.1 Comparing Options within STOP

Based on the options for setting selection, machine learning, and number of settings to try, there are an infinite number of ways to run STOP. In this section, we informally compare 21 configurations for running STOP, all of them based on testing a total of 64 settings. We have 3 options for selection of settings: random, pairwise, heuristic. We tested 7 different options for machine learning: no machine learning with $64 + 0$ (the first number in the addition represents the number of initial settings, and the second one is the number of additional settings using machine learning), regression trees with $48 + 16$, with $32 + 32$, with $16 + 48$, neural networks with $48 + 16$, with $32 + 32$, with $16 + 48$.

Table 6.3 shows the solution times of the tests on misc instances with CPLEX with six parameters. This is a single example for illustration and is not meant to be a conclusive comparison. The default setting takes 81.84 seconds. On the other hand, the solution time

Table 6.3: Best run time found by 21 different configurations of STOP for CPLEX on misc instances.

	Random	Pairwise	Heuristic
No learning 64	41.36	41.37	39.95
Regression tree $48 + 16$	37.94	38.27	36.16
Regression tree $32 + 32$	37.02	38.02	36.05
Regression tree $16 + 48$	39.75	37.62	36.51
Neural net $48 + 16$	36.57	38.32	36.79
Neural net $32 + 32$	37.57	35.26	35.86
Neural net $16 + 48$	44.99	40.09	39.59

for the true best setting is 34.00 seconds. As can be seen from the Table 6.3, the results of all 21 options are similar. The longest one takes 44.99 seconds and the fastest one 35.26 seconds. Results of random selection are generally worse than the other two selection options. We believe the main reason for this is that random selection misses some parameter interaction. The best settings are often found when the greedy heuristic option is used.

In general, we found similar behavior in our computational tests. There is not a large difference among the options for selection of settings. However, the best settings are generally observed when pairwise coverage is used. Table 6.4 shows how often each configuration of STOP found the best setting (out of the 21 configurations for STOP). Out of 11 tests in which all 21 configurations were considered, pairwise coverage found the best setting in 6 of them. The random option found the best setting in 3 tests, whereas the heuristic option found the best in 2 tests.

Table 6.4: Number of times that each configuration was the best out of the 21 configurations.

	Random	Pairwise	Heuristic
No learning 64		1	1
Regression tree $48 + 16$			
Regression tree $32 + 32$			
Regression tree $16 + 48$	2		
Neural net $48 + 16$		2	
Neural net $32 + 32$		2	1
Neural net $16 + 48$	1	1	

Machine learning helps to find the best setting. In 9 out of the 11 tests, the best settings were observed when one of the machine learning options is used. Neural networks were used when finding the best setting in 7 tests, whereas regression trees were used when finding the best setting in 2 tests. For the remaining 2 tests the best setting is observed when machine learning is not used.

Based on this preliminary analysis, we focus our tests on two of the configurations, both using pairwise coverage and a neural network. We refer to these as $p32nn32$ (for the $32 + 32$ option) and $p48nn16$ (for the $48 + 16$ option).

6.6.2 Comparison with Default Settings

In this section, we compare the solution times of the settings obtained from STOP with the times of the default settings of the MILP solvers. Based on the analysis from the previous section, we focus our attention on two possible options for STOP: $p32nn32$ and $p48nn16$. The p refers to the choice of pairwise coverage, the nn refers to the use of a neural network, while the first and second numbers indicate how many settings are tested before and after machine learning is performed, respectively.

Table 6.5: Solution times comparison on CPLEX. Note that "Worst" and "Best" refer to the setting reported by STOP with the worst and best of the 21 configurations.

CPLEX		Default (s)	STOP Methods (% improvement) Worst	p32nn32	p48nn16	Best	True best
p	6 par.	0.6059	32.9%	34.3%	35.2%	35.2%	36.0%
	10 par.	0.6109	33.0%	35.2%	36.2%	36.6%	41.6%
misc	6 par.	81.84	45.0%	56.9%	53.2%	56.9%	58.5%
	10 par.	81.41	36.4%	40.7%	47.3%	58.5%	unknown
mapping	6 par.	98.84	2.4%	35.4%	33.5%	35.9%	unknown
aircraft-med	6 par.	505.79	60.7%	69.7%	66.2%	69.8%	unknown
aircraft-easy	6 par.	1.62	35.2%	54.9%	54.3%	55.6%	unknown

In Table 6.5, Table 6.6 and Table 6.7, we show the solution times for the default settings and percentage improvement of STOP settings found with $p32nn32$ and $p48nn16$. In cases where we tested all 21 configurations for STOP, we also report the percentage improvement for the worst of the 21 configurations, and the percentage improvement for the best of the 21 configurations. To demonstrate the potential usefulness of STOP, however, we believe that it makes more sense to look at the performance of a single option, such as $p32nn32$ or $p48nn16$, across all solvers and instance classes.

For p and misc instances all 1296 ($= 3^4 * 4^2$) settings were tested for CPLEX with six parameters to find the true best solution time. For p instances, we also tested all 104976 settings for CPLEX with ten parameters. We did these exhaustive tests in order to see how STOP compares to the true best.

As can be seen from Table 6.5, Table 6.6 and Table 6.7, STOP's worst method's settings are always better than the default settings. Moreover, the reference configurations of $p32nn32$ and $p48nn16$ find settings that are often much faster than the default settings. In only one case, mapping instances on CBC (Table 6.6), does STOP fail to find a significant

Table 6.6: Solution times comparison on CBC.

		STOP Methods (% improvement)			
CBC	Default (s)	Worst	p32nn32	p48nn16	Best
p	22.57	74.1%	74.5%	74.3%	74.5%
misc	129.37	52.6%	77.9%	82.7%	82.7%
mapping	58.76	1.0%	1.0%	3.0%	3.3%
aircraft-easy	137.28	87.7%	88.2%	87.7%	88.2%

Table 6.7: Solution times comparison on GLPK. Because of long running times, only two configurations of STOP were tested.

		STOP Methods (% improvement)	
GLPK	Default (s)	p32nn32	p48nn16
p	3.08	45.8%	30.8%
misc	533.71	87.2%	87.2%
mapping	593.26	63.0%	63.1%
aircraft-easy	2024.35	99.5%	99.4%

improvement over the default setting. However, the solution time for the default setting on CBC is faster than the best method of CPLEX using the 10 parameters we selected, 63.31 (Table 6.5). This indicates that the default CBC setting is good for the mapping instances, which is why STOP was not able to find a much better setting.

6.6.3 Tests on Similar Instances

In Section 6.2, we stated that an assumption for our approach is that good settings for the test instances will be good settings for similar type of instances. To test this assumption and to see how well STOP behaves, we used the settings reported from STOP to test similar instances.

We considered three instances (sobel, laplace, decoder) to find better parameter settings for the mapping instances with cardinality 8 interconnect, three instances (sobel, laplace, idctrow) for the mapping instances with cardinality 5 interconnect and three instances (airland4-R2, airland5-R1, airland5-R2) for aircraft landing. We have different training sets for the mapping instances since we selected the three instances that can be solved fast. The default settings and the suggested settings of STOP can be seen in Tables 6.8, 6.9, and 6.10. We did not perform additional tests with p or misc instances since there are not many additional instances from those classes.

Tables 6.11, 6.12, 6.13, and 6.14 show the solution times of the default settings, STOP's suggested settings, and the ratios of the solution times of the default setting and STOP's suggested settings for the mapping instances. A ratio greater than 1 means STOP's suggested setting did better. We performed these tests with a one hour time limit. STOP's settings perform worse than the default settings for only decoder with cardinality 5 interconnect and the change is insignificant. In other cases for which an optimal solution was found, STOP's settings lead to a shorter solution time. The improvement is significant most of the time.

Tables 6.15, 6.16, and 6.17 show the solution times comparison between the default settings and STOP's settings for the aircraft landing instances. In only two cases did instances have longer solution times with the STOP settings, and in both these cases the difference is less than two seconds. In many cases, however, the improvement by STOP is also in-

Table 6.8: CPLEX's default setting and the suggested settings of STOP.

Parameter	Default	Mapping (card. 8)	Mapping (card. 5)	Aircraft-med
MIP emphasis	automatic	feasibility	feasibility	feasibility
Node selection	best-bound	best-bound	best-bound	best-bound
Branch. var. sel.	automatic	automatic	pseudo costs	pseudo costs
Dive type	automatic	traditional dive	probing dive	trad. dive
Fractional cuts	automatic	off	automatic	on
MIR cuts	automatic	off	on	automatic

Table 6.9: CBC default setting and the suggested settings of STOP.

Parameter	Default	Mapping (card. 8)	Aircraft-easy
Strong branching	5	0	0
Cost strategy	off	priorities	priorities
Gomory cuts	ifmove	off	root
MIR cuts	ifmove	ifmove	root
Probing	ifmove	off	on
Clique cuts	ifmove	on	root
Feasibility pump	on	on	off

Table 6.10: GLPK default setting and the suggested settings of STOP.

Parameter	Default	Mapping (card. 8)	Aircraft-easy
Scaling	equilibration	equilibration	equilibration
Pricing	steepest edge	steepest edge	largest coeff
Branching	driebeck-tomlin	most fractional	last var
Backtrack	best projection	best bound	breadth first
LP presolve	off	on	on
Method	new b&b	old b& b	new b& b with cuts

Table 6.11: (CPLEX) The solution times comparison between the default setting and STOP's settings for the mapping instances for cardinality 8 interconnect.

Instance	Default	STOP	Ratio
gsm	168.17	81.64	2.06
idctcol	578.57	358.24	1.62
idctrow	238.41	152.70	1.56
coder	386.90	243.62	1.59

Table 6.12: (CPLEX) The solution times comparison between the default setting and STOP's settings for the mapping instances for cardinality 5 interconnect.

Instance	Default	STOP	Ratio
gsm	3060.10	1590.34	1.92
idctcol	2558.75	548.18	4.67
decoder	2864.10	2943.01	0.97
coder	2731.29	1960.14	1.39

Table 6.13: (CBC) The solution times comparison between the default setting and STOP's settings for the mapping instances for cardinality 8 interconnect.

Instance	Default	STOP	Ratio
gsm	33.09	31.34	1.06
idctcol	3600	3600	1.00
idctrow	3456.58	2466.63	1.40
coder	277.67	16.61	16.72

Table 6.14: (GLPK) The solution times comparison between the default setting and STOP's settings for the mapping instances for cardinality 8 interconnect.

Instance	Default	STOP	Ratio
gsm	1568	661.61	2.37
idctcol	3600	3600	1.00
idctrow	891.33	696.81	1.28
coder	3600	3600	1.00

significant. However, when the difference is significant STOP does significantly better, such as with airland8-R2 with CPLEX and airland6-R2 with GLPK. The results for CBC are the least impressive, though even in this case the setting recommended by STOP is slightly better than the default.

6.7 TESTS ON MIPLIB AND MAPPING INSTANCES CONSIDERING DIFFERENT METRICS

In this section, we present results of tests of STOP on mapping and MIPLIB instances [2, 9]. For the MIPLIB instances, we find a different setting for each instance since there are not similar types of instances. With 64 settings, and a time limit of one hour, it could take up to 64 hours of CPU time to find such a setting. However, for the mapping instances, we use a single setting which is obtained from the training set. The purpose of the MIPLIB tests is to get a sense for the potential improvements, even though actual improvements across a family would not generally be so good. In contrast, experiments on families show both that families can benefit from the same parameter setting and show potential benefits for the entire family.

We compare the settings obtained from STOP with the CPLEX default setting. In addition to time-to-optimality, we will use two other metrics: proven gap and best-integer-solution.

- Proven Gap at time t: Proven gap is the absolute value of the ratio of the difference between the best integer objective and the LP bound, and the LP bound within time period t, that is $provengap = |\frac{incumbent-LPbound}{LPbound}|$.
- Best-Integer-Solution at time t: Best-integer-solution is the objective function value of the best integer feasible solution that has been found within time period t.

We use the configuration $p48nn16$ in STOP for the computational tests. The initial 48 settings are determined using pairwise coverage and the additional 16 settings are decided by neural networks.

Table 6.15: (CPLEX) The solution times comparison between the default setting and STOP's settings for the aircraft landing.

Instance	Default	STOP	Ratio
airland1-R1	0.05	0.04	1.25
airland2-R3	0.21	0.17	1.24
airland3-R2	0.61	1.44	0.42
airland4-R3	64.38	59.34	1.08
airland5-R3	93.52	76.63	1.22
airland6-R2	4.51	2.39	1.89
airland7-R1	0.58	0.56	1.04
airland8-R1	2.83	2.83	1.00
airland8-R2	14268.83	3372.61	4.23
airland8-R3	11.56	8.55	1.35

Table 6.16: (CBC) The solution times comparison between the default setting and STOP's settings for the aircraft landing.

Instance	Default	STOP	Ratio
airland1-R1	0.30	0.26	1.15
airland2-R3	9.96	4.34	2.29
airland3-R2	5.27	4.88	1.08
airland4-R3	3600	3600	1.00
airland5-R3	3600	3600	1.00
airland6-R2	11.16	12.70	0.88
airland7-R1	0.77	0.74	1.04
airland8-R1	453.44	393.38	1.15
airland8-R2	3600	3600	1.00
airland8-R3	3600	3600	1.00

Table 6.17: (GLPK) The solution times comparison between the default setting and STOP's settings for the aircraft landing.

Instance	Default	STOP	Ratio
airland1-R1	0.43	0.18	2.39
airland2-R3	1.48	0.86	1.72
airland3-R2	3600	4.23	$\geq$851.06
airland4-R3	3600	3600	1.00
airland5-R3	3600	3600	1.00
airland6-R2	3600	551.11	$\geq$ 6.53
airland7-R1	69.48	5.46	12.73
airland8-R1	3600	3600	1.00
airland8-R2	3600	3600	1.00
airland8-R3	620.05	505.99	1.23

Table 6.18: The solution times comparison between the default setting and STOP's settings for mapping instances.

Instance	Default	STOP	Ratio
Gsm	3060.10	1590.34	1.92
Decoder	2864.10	2943.01	0.97
Encoder	2731.29	1960.14	1.39
Idctcol	2558.75	548.18	4.67
Idctrow	1886.82	1385.01	1.36

6.7.1 Tests on Time-to-Optimality

This section considers the mapping and MIPLIB instances that take between 10 seconds and 10800 seconds. Tables 6.18 and 6.19 show the solution times of the default setting, STOP's suggested settings and the ratios of the solution times of the default setting and STOP's suggested settings. We use a time limit of 1 hour in STOP. The mapping instances are, on the average, 2.06x faster. For the MIPLIB instances, if we ignore 2 instances—swath, noswot—which cannot be solved in an hour, 12 out of 21 instances can be solved more than 2x faster. Six of the instances—opt1217, arki001, fast0507, glass4, manna81, harp2—which cannot be solved in an hour by the CPLEX default setting are solved with settings obtained from STOP, two of them—opt1217 and manna81—taking less than 1 second.

Similar to the discussion in Section 6.4, we analyzed the different settings for the instances "opt1217" and "manna81" to examine which parameters are critical. For "opt1217", only 6 settings—settings 1-6 in Table 6.20—are good out of 64 that were tested. In these 6 settings, two parameters seem critical: MIP emphasis and MIR cuts. MIP emphasis is at value 3 (moving best-bound) and MIR cuts is at value 1 (automatic). However, setting these two parameters at their best value is not enough. For two out of 64 settings—settings 7 and 8 in Table 6.20—CPLEX cannot solve "opt1217" within 1 hour even if these parameters are

Table 6.19: The solution times comparison between the default setting and STOP's settings for MIPLIB instances.

Instance	Default	STOP	Ratio
aflow30a	93.22	39.63	2.35
air04	28.74	21.65	1.33
air05	23.02	13.94	1.65
disctom	370.5	129.37	2.86
mas76	62.17	47.41	1.31
misc07	62.86	32.88	1.91
mod011	120.03	96.00	1.25
mzzv42z	173.35	83.61	2.07
nw04	27.62	4.59	6.02
pk1	81.57	67.46	1.21
qiu	207.18	81.88	2.53
rout	369.29	100.68	3.67
stein45	15.79	14.54	1.09
mas74	1482.52	1295.64	1.14
mzzv11	636.84	505.00	1.26
opt1217	> 3600	0.26	> 13846.15
swath	> 3600	> 3600	-
arki001	> 3600	21.55	> 167.05
fast0507	> 3600	1071.37	> 3.36
glass4	> 3600	463.25	> 7.77
manna81	> 3600	0.55	> 6545.45
harp2	> 3600	239.90	> 15.01
noswot	> 3600	> 3600	-

Table 6.20: Key CPLEX parameters for opt1217.

	Settings								
CPLEX (opt1217)	1	2	3	4	5	6	7	8	Default
MIP emphasis	3	3	3	3	3	3	3	3	0
Node selection	1	2	0	2	2	1	1	2	0
Branching var. sel.	2	0	1	2	1	0	0	2	0
Dive type	1	0	1	2	1	0	3	0	0
Fractional cuts	1	2	2	1	2	1	0	0	1
MIR cuts	1	1	1	1	1	1	1	1	1
Disjunctive cuts	0	1	1	2	0	0	0	2	1
Clique cuts	0	1	2	2	2	2	1	0	1
Node presolve sel.	0	0	2	1	1	0	0	1	1
Probing	1	1	2	0	1	2	2	2	1
Time (s)	0.296	0.284	0.380	0.260	0.292	0.292	> 3600	> 3600	> 3600

set correctly. Fractional cuts should not be at value 0 (off) to be able to solve the instance within 1 hour.

For "manna81" only 7 settings are good out of 64 that were tested. In these 7 settings, which can be seen in Table 6.21, MIP emphasis is at value 3 (moving best-bound) and fractional cuts is at value 1 (automatic). This combination leads to shorter solution times.

6.7.2 Tests on Proven Gap

There are 36 MIPLIB instances that take at least 600 seconds to solve and 34 instances that take at least 3600 seconds. This section presents results of tests of these instances on STOP with the proven gap metric. We do not test the mapping instances since the LP bound is 0 for all feasible instances and the proven gap values are undefined.

Tables 6.22 and 6.23 show the proven gaps using the CPLEX default setting, STOP's suggested settings and the ratios of the proven gaps of the default setting and STOP's suggested settings. "-" means no integer solution has been found yet. If the CPLEX default setting cannot find an integer solution and STOP finds an integer solution, we use a ratio of ∞. We also use a ratio of ∞ when STOP's setting finds the optimal objective value. The

Table 6.21: Key CPLEX parameters for manna81.

	Settings							
CPLEX (manna81)	1	2	3	4	5	6	7	Default
MIP emphasis	3	3	3	3	3	3	3	0
Node selection	2	2	1	0	1	1	0	0
Branching var. sel.	0	0	2	0	1	2	2	0
Dive type	1	2	3	2	3	1	2	0
Fractional cuts	1	1	1	1	1	1	1	1
MIR cuts	2	1	2	2	1	0	1	1
Disjunctive cuts	0	2	2	2	0	0	1	1
Clique cuts	0	1	2	1	1	2	0	1
Node presolve selector	1	1	2	0	1	0	2	1
Probing	0	1	2	2	0	1	0	1
Time (s)	0.568	0.564	0.572	0.576	0.568	0.556	0.572	> 3600

time limit used in STOP is 10 minutes for Table 6.22 and 1 hour for Table 6.23. The settings found are tied to the time limit.

As can be seen from Table 6.22, three of the instances—mzzv11, opt1217, manna81—are solved to optimality by STOP's settings while the CPLEX default setting cannot find the optimal solution within 10 minutes. STOP's suggested settings find integer solutions for 33 out of 36 instances whereas the CPLEX default setting can find integer solutions for 25 instances. In other words, STOP's recommended settings find integer solutions for 8 of the instances—swath, atlanta-ip, momentum2, msc98-ip, net12, protfold, rd-rdplusc-21, t1717—which do not have any integer solutions at 10 minutes with the CPLEX default setting.

In Table 6.23, 6 out of 34 instances—opt1217, fast0507, glass4, harp2, manna81, noswot—are solved to optimality by STOP's suggested settings while the CPLEX default setting cannot find the optimal solution within 1 hour. STOP's recommended settings find integer solutions for 32 instances whereas the CPLEX default setting can only find integer solutions for 25 instances. In other words, STOP's suggested settings find integer solutions for 7 of the instances—swath, atlanta-ip, momentum1, momentum2, msc98-ip, protfold, rd-rdplusc-21—which do not have any integer solutions at 3600 seconds with the CPLEX default setting.

Table 6.22: The proven gap comparison between the default setting and STOP's settings at 600 seconds.

Instance	Default	STOP	Ratio
mas74	0.02884	0.0231	1.25
mzzv11	0.00229	**0**	∞
opt1217	0.20	**0**	∞
swath	-	0.24105	∞
arki001	0.00013	0.00007	1.86
fast0507	0.02111	0.00905	2.33
glass4	1.125	0.90626	1.24
harp2	0.00127	0.00009	14.11
manna81	0.00454	**0**	∞
noswot	0.04651	0.04651	1.00
a1c1s1	0.1502	0.10443	1.44
aflow40b	0.077	0.05118	1.50
atlanta-ip	-	0.46423	∞
dano3mip	0.25664	0.22481	1.14
danoint	0.03928	0.0371	1.06
ds	10.2083	5.10792	2.00
liu	1.47143	1.13214	1.30
markshare1	-*	-*	-
markshare2	-*	-*	-
mkc	0.01429	0.01025	1.39
momentum1	-	-	-
momentum2	-	0.70047	∞
momentum3	-	-	-
msc98-ip	-	0.04746	∞
net12	-	2.1358	∞
nsrand-ipx	0.02215	0.01652	1.34
protfold	-	0.51183	∞
rd-rplusc-21	-	1673.53	∞
roll3000	0.06531	0.01557	4.19
seymour	0.04121	0.03344	1.23
sp97ar	0.02308	0.01487	1.55
stp3d	-	-	-
t1717	-	0.51446	∞
timtab1	0.49944	0.28495	1.75
timtab2	1.11536	0.97671	1.14
tr12-30	0.00258	0.00056	4.61

*LP bound is 0.

Thirty four instances are considered in both the 10-minute tests and the 1-hour tests. Compared to the CPLEX default setting running for an hour, STOP's suggested settings do as well or better in 10 minutes for 25 out of 34 MIPLIB instances which shows the importance of tuning.

6.7.3 Tests on Best-Integer-Solution

This section presents tests of mapping and MIPLIB instances on STOP with the best-integer-solution metric. The MIPLIB problems are minimization problems. Tables 6.24, 6.25 and 6.26 summarize the integer solutions found by the CPLEX default setting, by STOP's suggested settings, the ratios of the integer solutions of the default setting and STOP's suggested settings, and the optimal objective value for each instance. "-" means no integer solution has been found yet. "?" means optimal objective value is not known in the literature. In our tests, there are 8 instances like this. The time limit used in STOP is 10 minutes for Tables 6.24 and 6.25, and 1 hour for Table 6.26. If the CPLEX default setting cannot find an integer solution and STOP's suggested settings find an integer solution, we use a ratio of ∞. If the default setting finds the optimal integer solution, it is not included in the ratio calculation since the solution cannot be improved.

As can be seen from the tests on mapping instances in Table 6.24, the integer solutions found are much better when the settings obtained from STOP are used. The CPLEX default setting cannot find an integer solution for gsm in 10 minutes. We use the same setting for every instance.

As can be seen from Table 6.25, the CPLEX default setting finds the optimal integer solution for 5 out of 36 MIPLIB instances whereas STOP's suggested settings find the optimal objective value for 13 instances, including the 5 instances for which the CPLEX default setting finds the optimal integer solution. STOP's recommended settings find integer solutions for 32 out of 36 instances whereas the default setting can only find integer solutions for 25 instances. In other words, STOP's suggested settings find integer solutions for 7 of the instances—swath, atlanta-ip, momentum2, msc98-ip, protfold, rd-rdplusc-21, t1717—which do not have any integer solutions at 10 minutes with the CPLEX default setting.

Table 6.23: The proven gap comparison between the default setting and STOP's settings at 3600 seconds.

Instance	Default	STOP	Ratio
opt1217	0.20	**0**	∞
swath	-	0.21	∞
arki001	0.00013	0.00006	2.17
fast0507	0.01281	**0**	∞
glass4	1.125	**0**	∞
harp2	0.00058	**0**	∞
manna81	0.00442	**0**	∞
noswot	0.03831	**0**	∞
a1c1s1	0.13277	0.07375	1.80
aflow40b	0.04908	0.02848	1.72
atlanta-ip	-	0.11482	∞
dano3mip	0.21675	0.19951	1.09
danoint	0.03378	0.03043	1.11
ds	7.18731	4.59462	1.56
liu	1.31786	1.09643	1.20
markshare1	-*	-*	-
markshare2	-*	-*	-
mkc	0.01339	0.00237	5.65
momentum1	-	0.34922	∞
momentum2	-	0.29937	∞
momentum3	-	-	-
msc98-ip	-	0.00693	∞
net12	0.95164	0.67844	1.40
nsrand-ipx	0.01702	0.01016	1.68
protfold	-	0.41730	∞
rd-rplusc-21	-	1655.25	∞
roll3000	0.05119	0.01121	4.57
seymour	0.03219	0.02740	1.17
sp97ar	0.01109	0.01015	1.09
stp3d	-	-	-
t1717	0.80754	0.33999	2.38
timtab1	0.33962	0.21648	1.57
timtab2	0.93077	0.78961	1.18
tr12-30	0.00136	0.0001	13.60

*LP bound is 0.

Table 6.24: The best-integer-solution comparison between the default setting and STOP's settings at 600 seconds for mapping instances.

Instance	Default	STOP	Ratio	Optimal Obj.
Gsm	-	6	∞	0
Decoder	6	3	2.00	0
Encoder	162	2	81	0
Idctcol	6	1	6.00	0
Idctrow	77	17	4.53	0

As can be seen from Table 6.26, the CPLEX default setting finds the optimal integer solution for 7 out of 34 instances whereas STOP's suggested settings find the optimal objective value for 16 instances. STOP's recommended settings find integer solutions for 32 out of 34 instances whereas the default setting can only find integer solutions for 25 instances. In other words, STOP's suggested settings find integer solutions for 7 of the instances—swath, atlanta-ip, momentum1, momentum2, msc98-ip, protfold, rd-rdplusc-21—which do not have any integer solutions at 1 hour with the CPLEX default setting.

Thirty four instances are considered in both the 10-minute tests and the 1-hour tests. Compared to the CPLEX default setting running for an hour, STOP's suggested settings do as well or better in 10 minutes for 32 out of 34 MIPLIB instances which shows the importance of tuning.

6.7.4 Metrics' Settings Comparison

This section compares the settings obtained from STOP for different metrics with each other. Thus, we will have an idea whether the type of the metric in tuning is important or not. In the computational tests, five settings are considered:

$S1$**:** Setting found by STOP within 1 hour when time-to-optimality metric is used.

Table 6.25: The best-integer-solution comparison between the default setting and STOP's settings at 600 seconds for MIPLIB instances.

Instance	Default	STOP	Ratio	Optimal Obj.
mas74	**11801.2**	**11801.2**	-	11801.2
mzzv11	-21688	**-21718**	1.00	-21718
opt1217	**-16**	**-16**	-	-16
swath	-	472.11	∞	467.04
arki001	7581490	7580900	1.00	7580810
fast0507	176	**174**	1.01	174
glass4	1700010000	**1200010000**	1.42	1200010000
harp2	-73882000	**-73899800**	1.00	-73899800
manna81	**-13164**	**-13164**	-	-13164
noswot	**-41**	**-41**	-	-41
a1c1s1	11780.9	11647.9	1.01	11503.4
aflow40b	1179	**1168**	1.01	1168
atlanta-ip	-	96.0099	∞	90.0099
dano3mip	725.067	706.267	1.03	?
danoint	**65.667**	**65.667**	-	65.6667
ds	654.76	353.768	1.85	?
liu	1382	1194	1.16	?
markshare1	15	4	3.75	1
markshare2	34	11	3.09	1
mkc	-562.492	-563.046	1.00	-563.846
momentum1	-	-	-	109143
momentum2	-	15414	∞	12314.2
momentum3	-	-	-	?
msc98-ip	-	-	-	19839500
net12	-	**214**	∞	214
nsrand-ipx	51680	**51200**	1.01	51200
protfold	-	-19	∞	-31
rd-rplusc-21	-	166592	∞	165395
roll3000	13125	12913	1.02	12890
seymour	428	425	1.01	423
sp97ar	669575000	664540000	1.01	?
stp3d	-	-	-	?
t1717	-	184777	∞	?
timtab1	774858	764774	1.01	764772
timtab2	1347050	1181330	1.14	?
tr12-30	130707	**130596**	1.00	130596

Table 6.26: The best-integer-solution comparison between the default setting and STOP's settings at 3600 seconds for MIPLIB instances.

Instance	Default	STOP	Ratio	Optimal Obj.
opt1217	**-16**	**-16**	-	-16
swath	-	468.485	∞	467.04
arki001	7581490	**7580810**	1.00	7580810
fast0507	175	**174**	1.01	174
glass4	1700010000	**1200010000**	1.42	1200010000
harp2	**-73899800**	**-73899800**	-	-73899800
manna81	**-13164**	**-13164**	-	-13164
noswot	**-41**	**-41**	-	-41
a1c1s1	11740.2	**11503.4**	1.02	11503.4
aflow40b	**1168**	**1168**	-	1168
atlanta-ip	-	93.0102	∞	90.0099
dano3mip	702.182	699.018	1.00	?
danoint	**65.6667**	**65.6667**	-	65.6667
ds	478.285	280.967	1.70	?
liu	1289	1158	1.11	?
markshare1	7	2	3.5	1
markshare2	21	9	2.33	1
mkc	-563.006	**-563.846**	1.00	-563.846
momentum1	-	128476	∞	109143
momentum2	-	13911.9	∞	12314.2
momentum3	-	-	-	?
msc98-ip	-	**19839500**	∞	19839500
net12	**214**	**214**	-	214
nsrand-ipx	51520	**51200**	1.01	51200
protfold	-	-23	∞	-31
rd-rplusc-21	-	166122	∞	165395
roll3000	12979	12893	1.01	12890
seymour	426	**423**	1.01	423
sp97ar	662999000	662278000	1.00	?
stp3d	-	-	-	?
t1717	244585	183237	1.33	?
timtab1	805690	764774	1.05	764772
timtab2	1275690	1133560	1.13	?
tr12-30	130655	**130596**	1.00	130596

$S2a$**:** Setting found by STOP within 10 minutes when proven gap metric is used.

$S2b$**:** Setting found by STOP within 1 hour when proven gap metric is used.

$S3a$**:** Setting found by STOP within 10 minutes when best-integer-solution metric is used.

$S3b$**:** Setting found by STOP within 1 hour when best-integer-solution metric is used.

For the mapping instances with cardinality 5 interconnect, we only compare the settings $S1$ and $S3a$ since the proven gap values are undefined. Table 6.27 shows the STOP's recommended settings—$S1$ and $S3a$. Four parameters have the same values: Branching variable selection, MIR cuts, Clique cuts and Node presolve selector.

Table 6.28 presents results of tests of STOP's suggested settings described in Table 6.27 when those settings are used on a metric other than the metric they are suggested for. For example, $S1$ is STOP's suggested setting for time-to-optimality metric at 1 hour. When we try this setting on the best-integer-solution metric for decoder, CPLEX finds the integer solution 27. Even though $S3a$ finds good integer solutions in 10 minutes for all instances, it cannot find the optimal solution for encoder and idctrow in an hour. Similarly, $S1$ finds the optimal solution faster, but it cannot find good integer solutions in 10 minutes.

In order to compare the performance of all of the settings—$S1$, $S2a$, $S2b$, $S3a$, $S3b$—we consider the six MIPLIB instances manna81, arki001, fast0507, glass4, harp2 and opt1217 which cannot be solved in an hour by the CPLEX default setting and can be solved in less than an hour by STOP's settings.

Table 6.29 presents the five suggested settings—$S1$, $S2a$, $S2b$, $S3a$, $S3b$—for manna81. Only the value of the first parameter—MIP emphasis (moving best-bound)—is the same for all settings. Table 6.30 shows results of tests of STOP's suggested settings described in Table 6.29 when those settings are used on metrics other than the metric they are suggested for. For example, when the setting $S3b$ is used, CPLEX cannot find the optimal solution within 1 hour.

As discussed in Section 6.7.1, the solution time for manna81 is short when the parameter MIP emphasis is at value 3 (moving best-bound) and the parameter fractional cuts is at value 1 (automatic). For settings $S3a$ and $S3b$, the parameter fractional cuts is not at its critical value, see Table 6.29. Therefore, settings obtained based on the best-integer-solution metric are not good for other metrics. In fact, all 64 settings tested by STOP with best-

Table 6.27: Settings comparison for mapping instances.

mapping	Metrics	
CPLEX	$S1$ (3600)	$S3a$ (600)
Parameters	Time-to-opt.	Best Int. Sol.
MIP emphasis	feasibility	automatic
Node selection	best-bound	alternative best-estimate
Branching var. sel.	pseudo costs	pseudo costs
Dive type	probing dive	traditional dive
Fractional cuts	automatic	off
MIR cuts	automatic	automatic
Disjunctive cuts	automatic	on
Clique cuts	automatic	automatic
Node presolve selector	automatic	automatic
Probing	automatic	on

Table 6.28: Settings on different metrics for mapping instances.

mapping		Metrics	
CPLEX			600 seconds
Instance	Setting	Time-to-opt.	Best Int. Sol.
Gsm	$S1$	1590.34	-
	$S3a$	2228.31	6
	Default	3060.10	-
Decoder	$S1$	2943.01	27
	$S3a$	651.53	3
	Default	2864.10	6
Encoder	$S1$	1960.14	162
	$S3a$	> 3600	2
	Default	2731.29	162
Idctcol	$S1$	548.18	0
	$S3a$	1384.62	1
	Default	2558.75	6
Idctrow	$S1$	1385.01	26
	$S3a$	> 3600	17
	Default	1886.82	77

Table 6.29: Settings comparison for manna81.

manna81	Metrics				
CPLEX	$S1$ (3600)	$S2a$ (600)	$S2b$ (3600)	$S3a$ (600)	$S3b$ (3600)
Parameters	Time-to-opt.	Proven Gap	Proven Gap	Best Int. Sol.	Best Int. Sol.
MIP emphasis	mov. best b.	mov. best b.	mov. best b.	mov. best b.	mov. best b.
Node selection	best-estimate	alt. best-est.	best-bound	best-estimate	best-estimate
Bran. var. sel.	strong bran.	automatic	strong bran.	automatic	strong bran.
Dive type	trad. dive	probing dive	guided dive	automatic	automatic
Fract. cuts	automatic	automatic	automatic	on	off
MIR cuts	off	off	automatic	on	on
Disj. cuts	off	on	automatic	automatic	on
Clique cuts	on	automatic	automatic	on	on
Node pre. sel.	off	force	off	off	off
Probing	automatic	on	on	off	automatic

Table 6.30: Settings on different metrics for manna81.

manna81	Metrics				
CPLEX		600 seconds	3600 seconds	600 seconds	3600 seconds
Settings	Time-to-opt.	Proven Gap	Proven Gap	Best Int. Sol.	Best Int. Sol.
$S1$	0.55	0	0	-13164	-13164
$S2a$	0.55	0	0	-13164	-13164
$S2b$	0.58	0	0	-13164	-13164
$S3a$	> 3600	0.00716	0.00716	-13164	-13164
$S3b$	> 3600	0.00948	0.00936	-13164	-13164
Default	> 3600	0.00454	0.00442	-13164	-13614

Table 6.31: Settings comparison for fast0507.

fast0507	Metrics				
CPLEX	$S1$ (3600)	$S2a$ (600)	$S2b$ (3600)	$S3a$ (600)	$S3b$ (3600)
Parameters	Time-to-opt.	Proven Gap	Proven Gap	Best Int. Sol.	Best Int. Sol.
MIP emphasis	automatic	optimality	feasibility	automatic	optimality
Node selection	best-estimate	alt. best-est.	alt. best-est.	alt. best-est.	best-estimate
Bran. var. sel.	pseudo costs	strong bran.	pseudo costs	pseudo costs	automatic
Dive type	guided dive	guided dive	automatic	automatic	guided dive
Fractional cuts	off	off	off	off	off
MIR cuts	automatic	on	off	on	automatic
Disj. cuts	on	off	on	on	off
Clique cuts	on	automatic	on	off	off
Node pre. sel.	force	automatic	force	automatic	force
Probing	off	off	off	on	automatic

integer-solution metric when finding $S3a$ or $S3b$ give the same integer solution, -13164. It is easy to find an integer solution for manna81, but hard to prove that it is the optimal solution. On the other hand, settings $S1$, $S2a$ and $S2b$ perform well for all metrics. They are better than the default setting for all scenarios.

Table 6.31 shows the five settings—$S1$, $S2a$, $S2b$, $S3a$, $S3b$—obtained from STOP for fast0507. Only the value (off) of the parameter fractional cuts is the same for all settings. Table 6.32 presents results of tests of STOP's suggested settings described in Table 6.31 when those settings are used on metrics other than the metric they are suggested for. All settings perform well with other metrics. They are better than the CPLEX default setting and find the optimal solution within 2000 seconds.

Table 6.33 presents the five settings—$S1$, $S2a$, $S2b$, $S3a$, $S3b$—obtained from STOP for arki001. The parameters node selection and fractional cuts are at values best-estimate and automatic for all settings, respectively.

Table 6.34 presents results of tests of STOP's suggested settings described in Table 6.33 when those settings are used on metrics other than the metric they are suggested for. All settings perform better than the default setting. They find the optimal solution within 650 seconds. In fact, all settings except $S3a$ solve arki001 within 90 seconds. Setting $S3a$ takes

Table 6.32: Settings on different metrics for fast0507.

fast0507	Metrics				
CPLEX Settings	Time-to-opt.	600 seconds Proven Gap	3600 seconds Proven Gap	600 seconds Best Int. Sol.	3600 seconds Best Int. Sol.
$S1$	1071.37	0.01520	0	175	174
$S2a$	1729.73	0.00875	0	174	174
$S2b$	1931.66	0.00993	0	174	174
$S3a$	1947.25	0.00993	0	174	174
$S3b$	1227.74	0.01520	0	175	174
Default	> 3600	0.02111	0.01281	176	175

Table 6.33: Settings comparison for arki001.

arki001	Metrics				
CPLEX Parameters	$S1$ (3600) Time-to-opt.	$S2a$ (600) Proven Gap	$S2b$ (3600) Proven Gap	$S3a$ (600) Best Int. Sol.	$S3b$ (3600) Best Int. Sol.
MIP emphasis	optimality	mov. best b.	mov. best b.	feasibility	optimality
Node selection	best-estimate	best-estimate	best-estimate	best-estimate	best-estimate
Bran. var. sel.	strong bran.	strong bran.	strong bran.	pseudo costs	automatic
Dive type	automatic	probing dive	guided dive	probing dive	automatic
Fractional cuts	automatic	automatic	automatic	automatic	automatic
MIR cuts	off	off	on	automatic	on
Disj. cuts	automatic	on	on	automatic	on
Clique cuts	on	off	on	automatic	automatic
Node pre. sel.	force	force	force	off	force
Probing	on	automatic	on	off	automatic

Table 6.34: Settings on different metrics for arki001.

arki001	Metrics				
CPLEX Settings	Time-to-opt.	600 seconds Proven Gap	3600 seconds Proven Gap	600 seconds Best Int. Sol.	3600 seconds Best Int. Sol.
$S1$	21.55	0	0	7580810*	7580810*
$S2a$	82.05	0.00007*	0.00007*	7580950*	7580950*
$S2b$	16.38	0.00008*	0.00006*	7581170*	7581170*
$S3a$	642.08	0.00010	0.00009*	7581490	7580810*
$S3b$	19.56	0.00009*	0.00009*	7580810*	7580810*
Default	> 3600	0.00013	0.00013	7581490	7581490

*Optimal ($mipgap = 0.0001$).

longer because the parameter node presolve selector is not at value force. The default relative mipgap tolerance (a relative tolerance on the gap between the best integer objective and the objective of the best node remaining) is 10^{-4} for CPLEX, therefore the optimal objective values are different for some cases for arki001.

Tables 6.35, 6.36 and 6.37 present results of tests of STOP's suggested settings used on metrics other than the metric they are suggested for. $S1$, $S2a$, $S2b$ and $S3b$ settings for glass, harp2 and op1217 always perform better than the CPLEX default setting. $S3a$ settings perform better most of the time.

Based on these preliminary tests, different metrics generally lead to different parameters. Sometimes this difference is important. If it is hard to find an integer solution for the instance, STOP should be used with the best-integer-solution metric. However, if finding an integer solution is easy or finding the optimal solution is essential, STOP should be used with the proven gap metric or time-to-optimality metric.

Table 6.35: Settings on different metrics for glass4.

glass4	Metrics				
CPLEX Settings	Time-to-opt.	600 seconds Proven Gap	3600 seconds Proven Gap	600 seconds Best Int. Sol.	3600 seconds Best Int. Sol.
$S1$	463.25	0	0	1200010000	1200010000
$S2a$	> 3600	0.90626	0.90626	1525010000	1525010000
$S2b$	2456.65	0.83334	0	1466680000	1200010000
$S3a$	> 3600	0.50000	0.50000	1200010000	1200010000
$S3b$	3341.18	0.95834	0	1800020000	1200010000
Default	> 3600	1.125	1.125	1700010000	1700010000

Table 6.36: Settings on different metrics for harp2.

harp2	Metrics				
CPLEX Settings	Time-to-opt.	600 seconds Proven Gap	3600 seconds Proven Gap	600 seconds Best Int. Sol.	3600 seconds Best Int. Sol.
$S1$	239.90	0	0	-73899800	-73899800
$S2a$	406.45	0	0	-73899800	-73899800
$S2b$	319.47	0	0	-73899800	-73899800
$S3a$	3560.33	0.00176	0	-73899800	-73899800
$S3b$	> 3600	0.00082	0.00028	-73899800	-73899800
Default	> 3600	0.00127	0.00058	-73882000	-73899800

Table 6.37: Settings on different metrics for opt1217.

opt1217	Metrics				
CPLEX Settings	Time-to-opt.	600 seconds Proven Gap	3600 seconds Proven Gap	600 seconds Best Int. Sol.	3600 seconds Best Int. Sol.
$S1$	0.26	0	0	-16	-16
$S2a$	0.36	0	0	-16	-16
$S2b$	0.38	0	0	-16	-16
$S3a$	> 3600	0.11111	0.11111	-16	-16
$S3b$	> 3600	0.15789	0.15789	-16	-16
Default	> 3600	0.20	0.20	-16	-16

6.8 CONCLUSION

We have presented a method to tune software parameters using ideas from software testing and machine learning. The method is based on the key observation that for many classes of instances, results will be good if a few critical parameters have good values, though the set of critical parameters depends on the class of instances.

Based on the computational tests, none of the three options for selecting settings appears to dominate the others, but the best settings are often observed when pairwise coverage is used. Machine learning certainly helps to find the best setting. The two reference configurations of $p32nn32$ and $p48nn16$ appear to be good choices.

In our computational tests of MILP solvers, the worst result our implementation reported out of 21 configurations is always better than the solution time of the default setting. Moreover, the solution times achieved by two reference configurations—$p32nn32$, $p48nn16$—are often much faster than the solution time of the default setting. The tests of related instances support the assumption that good settings for the test instances will also be good for similar instances. For example, mapping instances are, on the average, 2.24x faster for CPLEX.

When settings obtained from STOP with time-to-optimality metric are used, 12 out of 21 MIPLIB instances can be solved more than 2x faster. Six of the instances which cannot be solved in an hour by the CPLEX default setting are solved with settings obtained from STOP. The mapping instances are, on the average, 2.06x faster. Six out of 34 instances are solved to optimality by STOP's recommended settings for the proven gap metric while the CPLEX default setting cannot find the optimal solution within 1 hour. The CPLEX default setting finds the optimal integer solution for 7 out of 34 MIPLIB instances whereas STOP's suggested settings with best-integer-solution metric find the optimal objective value for 16 instances within 1 hour. These results show that there are generally settings much better than the default setting.

Although developed for MILP solvers, our method is flexible and may be applicable to other algorithms for which the key observation is valid. In addition, the method can be easily extended and may be useful with different metrics, such as the maximum solution time for several instances, or the solution quality for a heuristic algorithm.

Our implementation of the method, STOP, has been released under a free and open-source license.

7.0 CONCLUSIONS AND FUTURE WORK

7.1 CONCLUSIONS

This dissertation focuses on a mapping problem required for the effective use of our low-energy computational fabric. Programming the fabric requires mapping the operators and connections of the desired data flow graph onto the ALUs and interconnect of the fabric. We have also considered the automated tuning of optimization software parameters which helps software users identify good parameter values for their instances.

In Chapter 4, we formulate two mixed integer linear programs—IP-Fixed and IP-General—to solve the various forms of the mapping problem. We present the modeling and algorithmic aspects that are necessary to make the IP-Fixed formulation competitive. The improved model can solve all the benchmarks for cardinality 5 interconnect in less than an hour. The IP-General formulation has not been the main focus of research due to the difficulty of solving it. Other researchers in our group consider different approaches for the mapping problem. Among these approaches, MILP solutions are important because they provide a reference for how good a mapping is possible.

In Chapter 5, we introduce the sliding partial MILP heuristic to solve the Augmented Fixed Rows problem. The sliding heuristic generates mappings faster than the IP-Fixed formulation and finds mappings for the instances that cannot be solved by the IP-Fixed formulation. However, the sliding heuristic may add rows when it is not strictly necessary. The sliding heuristic can also be used with other approaches. Using the heuristic as a post-processor makes SA usable since SA cannot find complete solutions to some instances on its own. When the sliding partial MILP heuristic does not add any rows of pass-gates, we know that the solution found is feasible for the IP-Fixed formulation. In other words, the

mappings generated by the IP-Fixed model and the sliding heuristic have the same quality. Thus, we show that sometimes intractable MILPs can be partitioned into smaller MILPs and can be solved faster. We also compare the sliding partial MILP heuristic with a sophisticated greedy heuristic that was created by other researchers on our team. The sliding partial MILP heuristic adds fewer rows than the greedy heuristic. In addition, the total path length is shorter when the sliding partial MILP heuristic is used. However, the sliding partial MILP heuristic takes more time.

In Chapter 6, we introduce a method to tune software parameters using ideas from software testing and machine learning. The method is based on the key observation that for many classes of instances, results will be good if a few critical parameters have good values, though the set of critical parameters depends on the class of instances.

In our computational tests of STOP, the solution times achieved by two reference configurations ($p32nn32$, $p48nn16$) are often much faster than the solution time of the default setting. The tests of related instances support the assumption that good settings for the test instances will also be good for similar instances. For example, mapping instances are, on the average, 2.24x faster for CPLEX. When settings obtained from STOP with time-to-optimality metric are used, 12 out of 21 MIPLIB instances can be solved more than 2x faster. Six of the instances which cannot be solved in an hour by the CPLEX default setting are solved with settings obtained from STOP. Compared to the CPLEX default setting running for an hour, STOP's suggested settings for the proven-gap metric do as well or better in 10 minutes for 25 out of 34 MIPLIB instances. Similarly, compared to the CPLEX default setting running for an hour, STOP's suggested settings for the best-integer-solution metric do as well or better in 10 minutes for 32 out of 34 MIPLIB instances.

These results show the importance of tuning parameters when the computational time investment is worthwhile. In particular, researchers working with families of instances such as the mapping instances may benefit significantly.

7.2 FUTURE WORK

Finding good mappings is the central theme of this research. Future work will focus on generating better mappings and testing larger instances.

The Feasible Mapping with Fixed Rows problem may be viewed as a special case of subgraph isomorphism. Subgraph isomorphism is NP-hard and future work may analyze whether our problems are also NP-hard.

The IP-General formulation has potential for improved performance with additional study. For example, it may be possible to tighten the MILP further or attempt to take advantage of the network structure when solving the LP relaxations by using a network version of the simplex algorithm. This may strengthen the LP relaxation and make it faster.

In the sliding partial MILP heuristic, we consider adjusting the columns of the operators and adding rows of pass-gates to fix violations. Another way to fix violations is changing the rows of individual operators. That is, moving the operators up or down instead of assigning them to certain rows. In order to do this, we need to know whether the violated edge is on the critical path or not. If it is not on the critical path, we can move the operators of the violated edge up or down depending on the slack of the operator. If the operator has a negative slack, we can move it up. if it has a positive slack, it can be moved down. On the other hand, if the violation is on the critical path, we need to add a row of pass-gates to fix the violation. Changing the rows of the operators may not always work, we may still need to add row of pass-gates to fix the violations. Future work may explore these possibilities.

The sliding partial MILP heuristic generates mappings by using small, fast MILPs. It gives an optimal solution like the IP-Fixed formulation if it does not add any rows of pass-gates. As a result, we show that sometimes large MILPs that take too long can be partitioned into smaller MILPs and can be solved faster. Future work will focus on applying this technique to optimization problems in different areas.

In STOP, the best settings are often observed when pairwise coverage is used, however none of the three options for selecting settings appears to dominate the others. We leave to future work a careful study of configuring STOP itself, including the possibility of a more sophisticated use of machine learning.

In this dissertation, STOP is tested with MILP solvers. Future work will focus on testing STOP with other types of softwares for which the key observation is valid. In addition, the method can be easily extended and may be useful with different metrics, such as the maximum solution time for several instances, or the solution quality for a heuristic algorithm.

APPENDIX

BENCHMARK INSTANCES

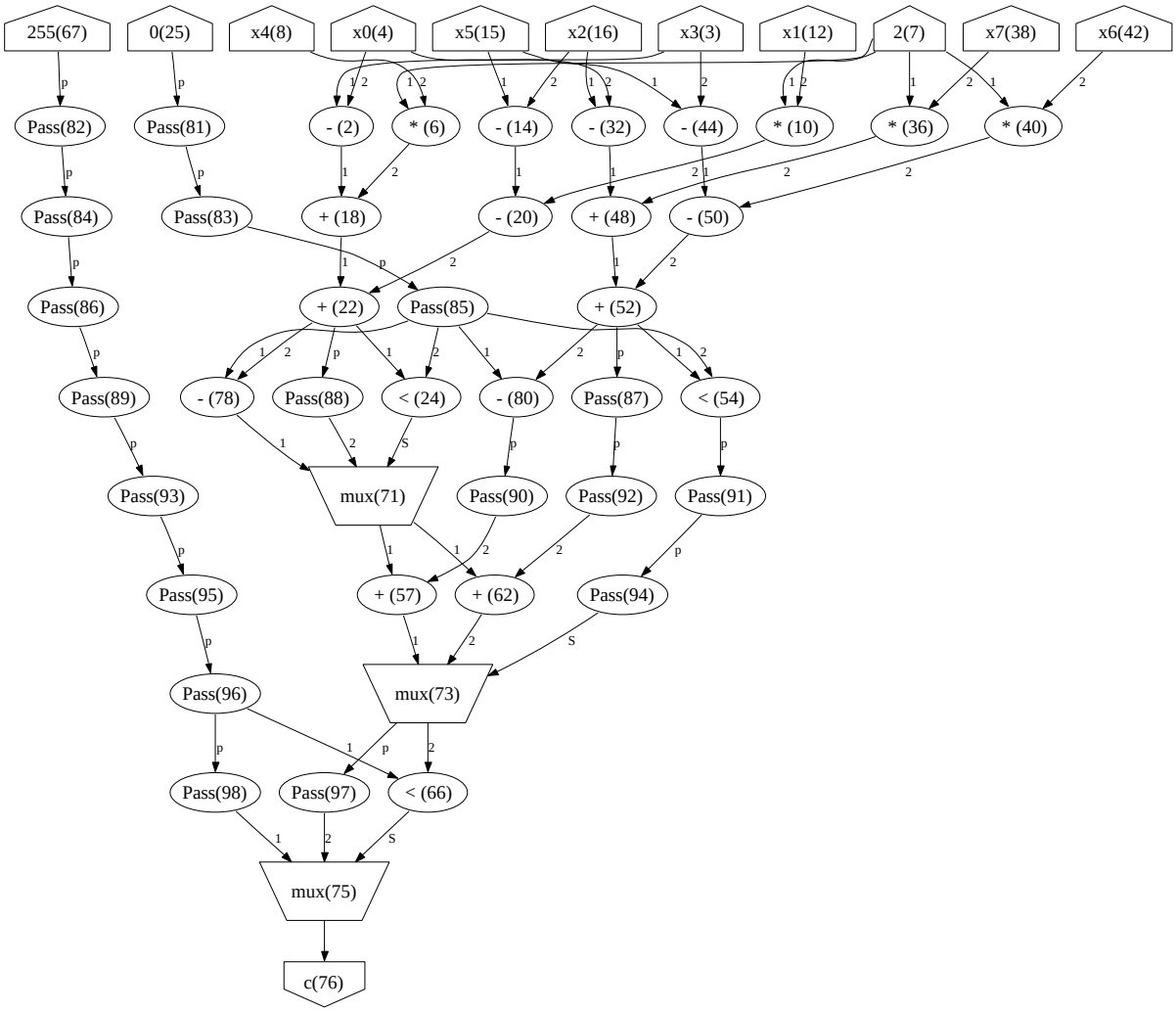


Figure A1: Sobel.

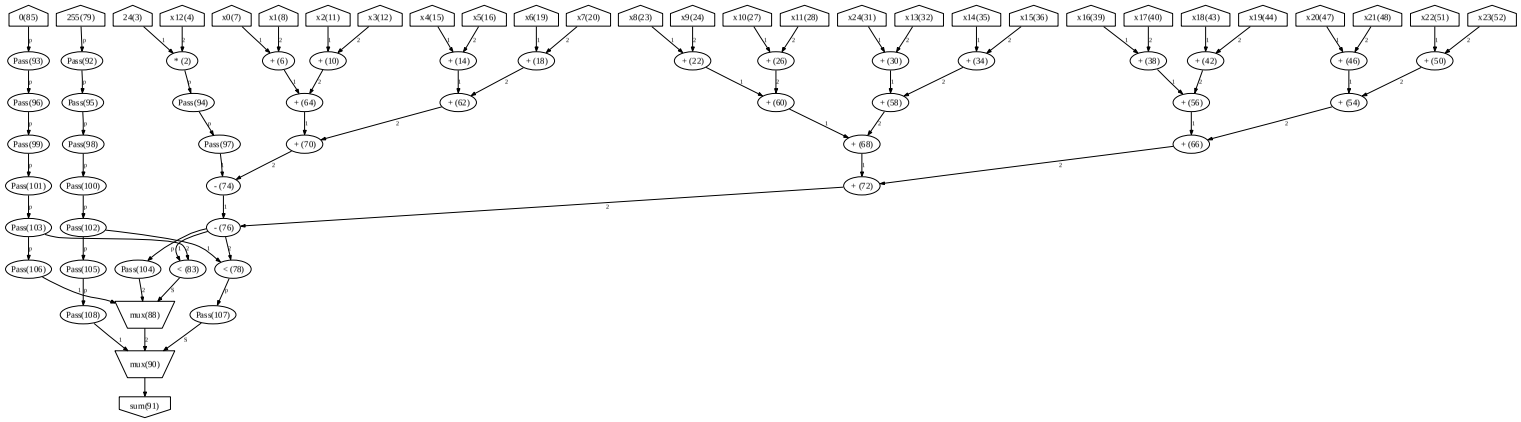


Figure A2: Laplace.

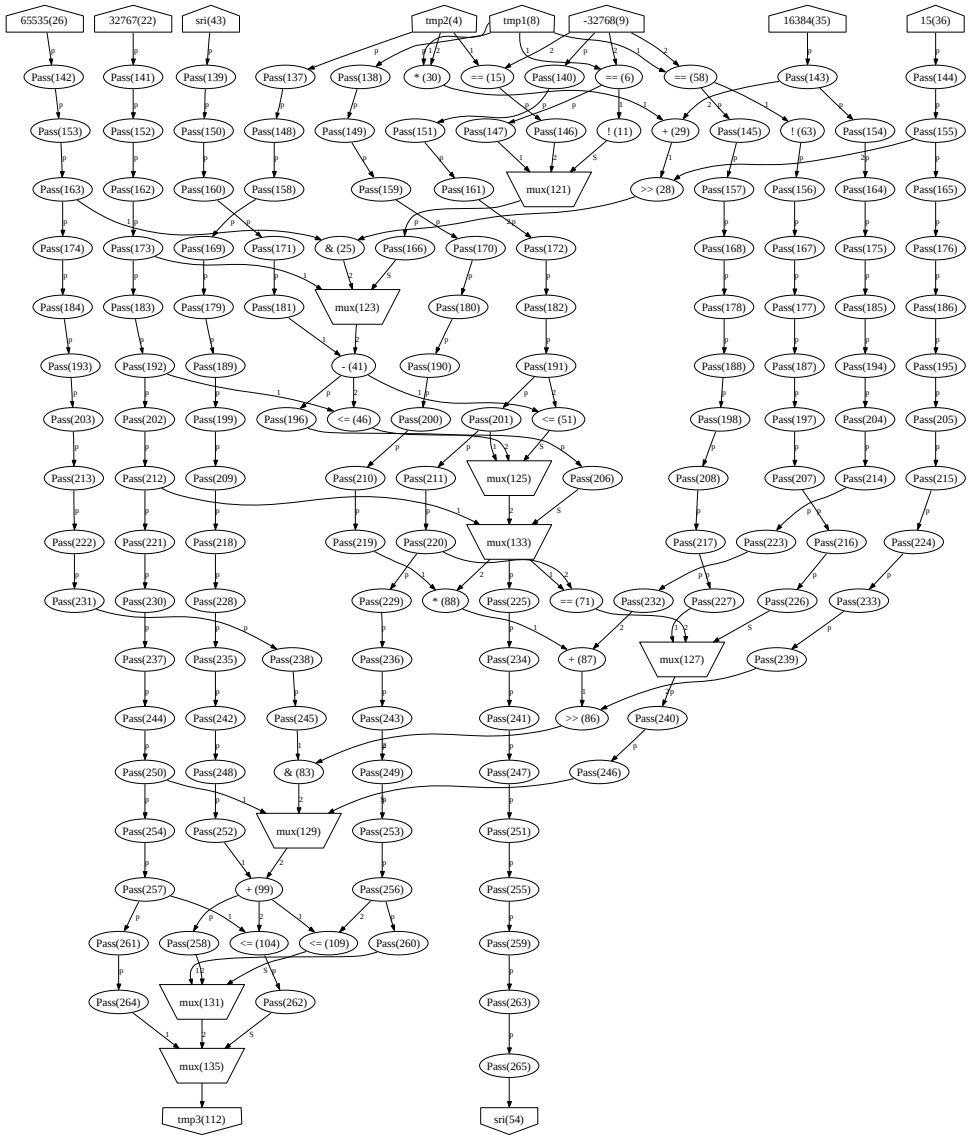


Figure A3: Gsm.

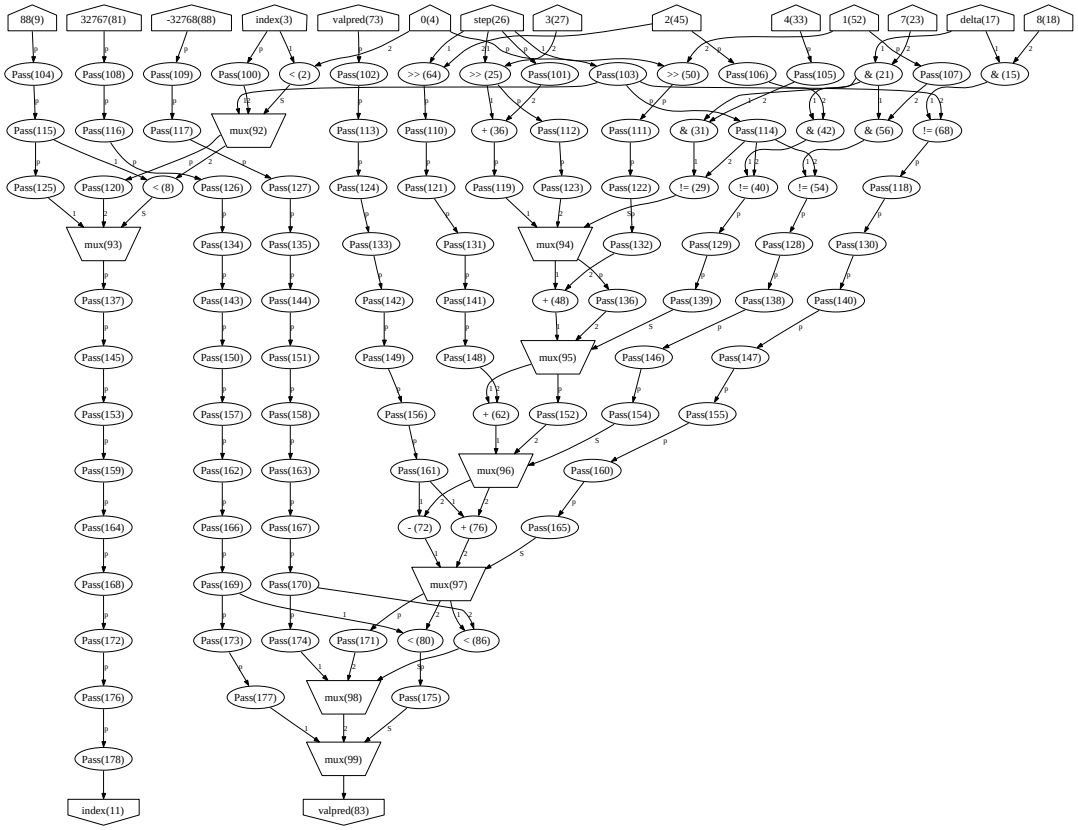


Figure A4: Decoder.

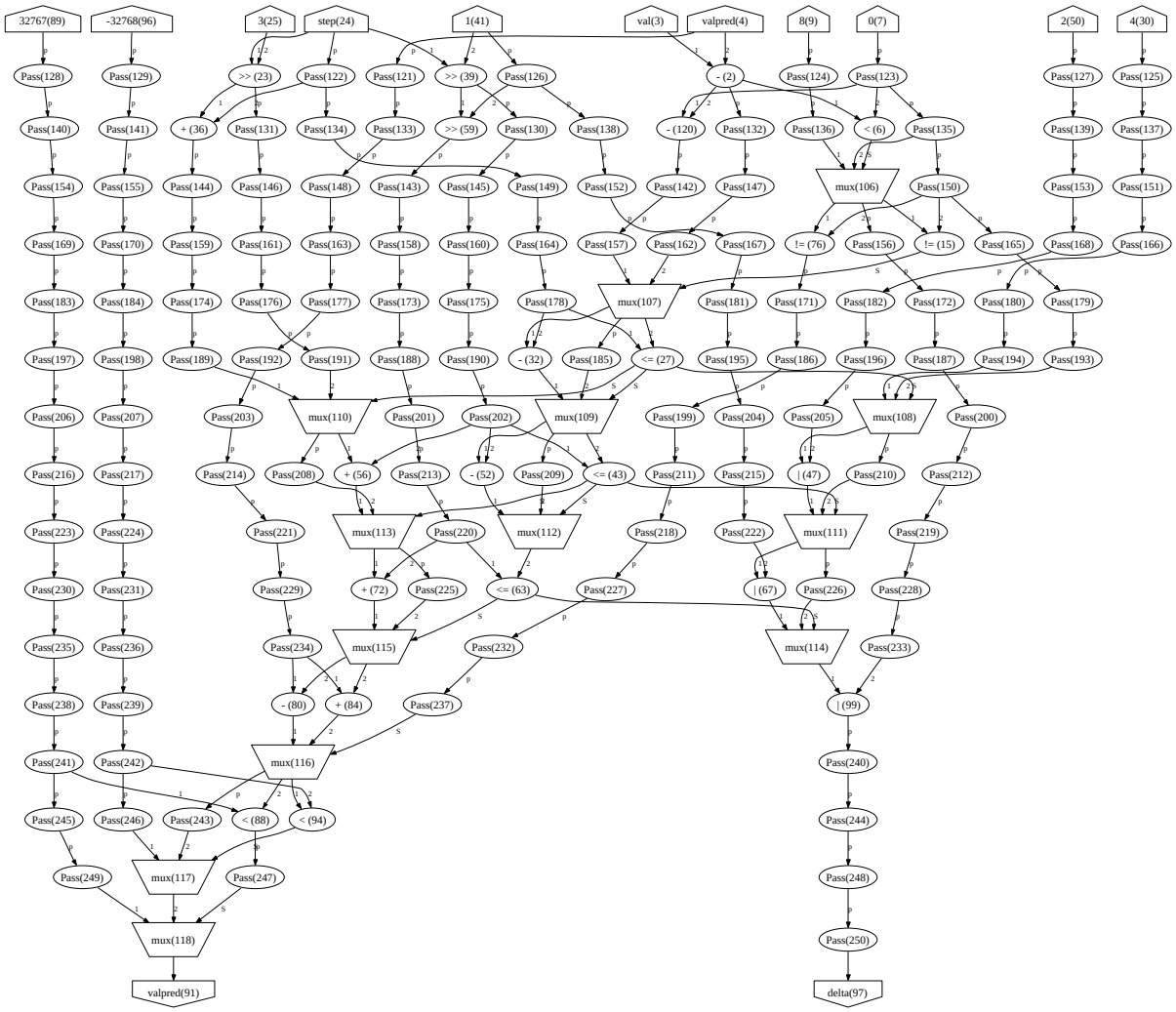


Figure A5: Coder.

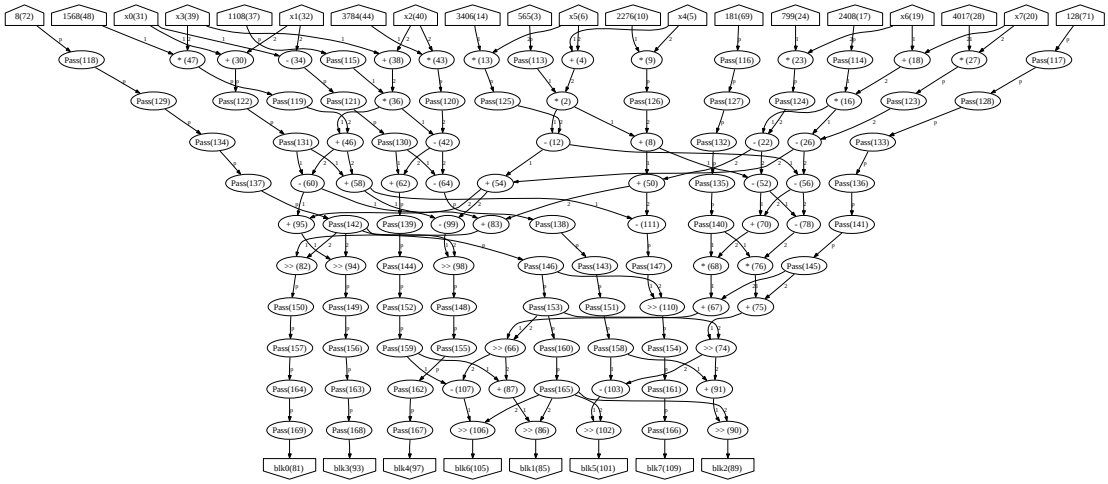


Figure A6: Idctrow.

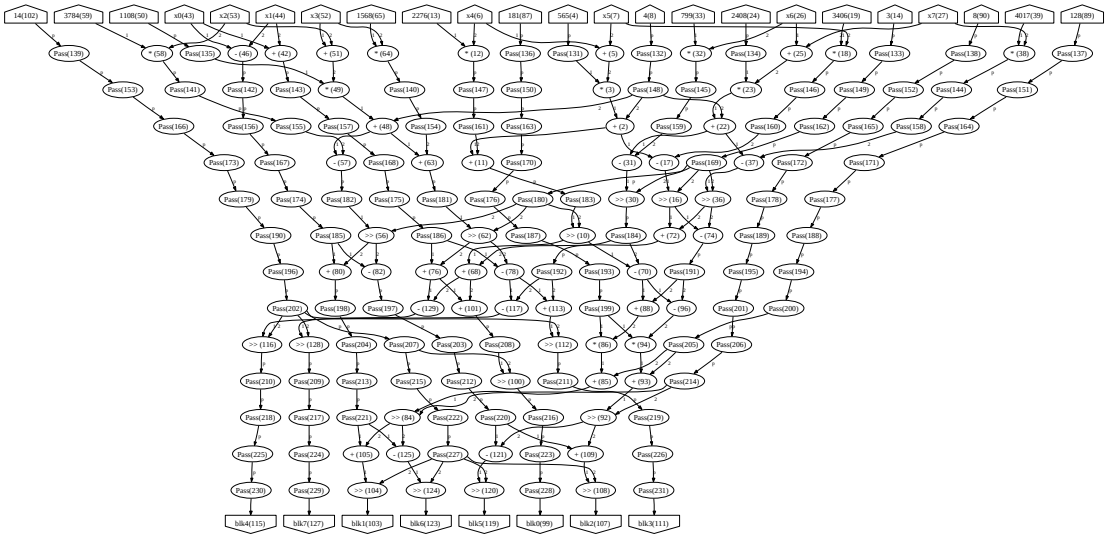
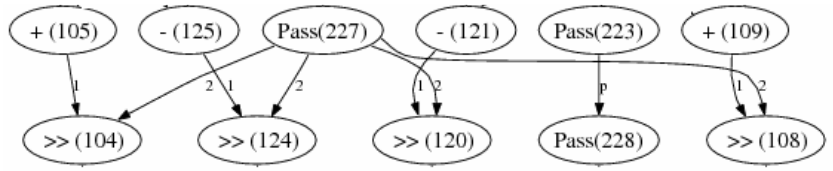
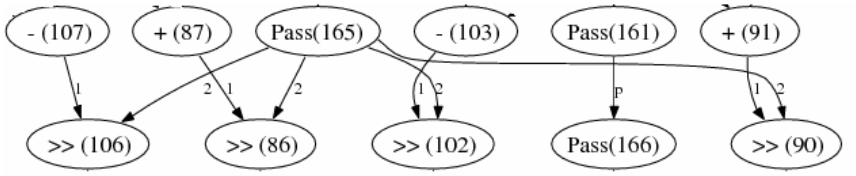


Figure A7: Idctcol.

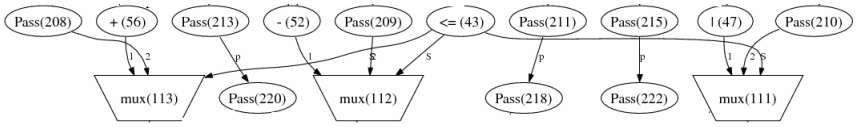
a) idctcol



b) idctrow



c) coder



d) sobel

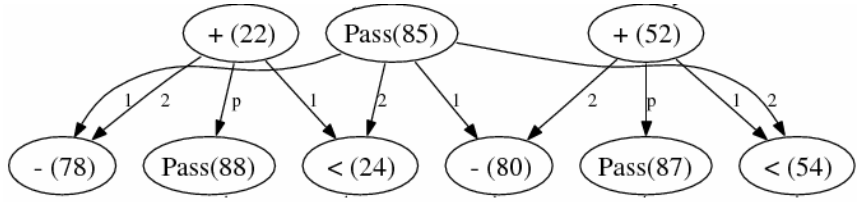


Figure A8: Infeasible structures.

BIBLIOGRAPHY

[1] P. Achard and E. De Schutter. Complex parameter landscape for a complex neuron model. *PLoS Comput Biol*, 2(7 e94), 2006.

[2] T. Achterberg, T. Koch, and A. Martin. MIPLIB 2003. *Operations Research Letters*, 34:1–12, 2006.

[3] Claudio Arbib and Fabrizio Marinelli. An optimization model for trim loss minimization in an automotive glass plant. *European Journal of Operational Research*, 127(3):1421–1432, December 2007.

[4] Charles Audet and Dominique Orban. Finding optimal algorithmic parameters using derivative-free optimization. *SIAM J. on Optimization*, 17(3):642–664, 2006.

[5] Francisco Barahona, Martin Grötschel, Michael Jünger, and Gerhard Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Oper. Res.*, 36(3):493–513, 1988.

[6] M. Baz, B. Hunsaker, P. Brooks, and A. Gosavi. Automated tuning of optimization software parameters. *University of Pittsburgh Department of Industrial Engineering Technical Report 2007-7, submitted to INFORMS Journal on Computing*, 2007.

[7] M. Baz, B. Hunsaker, G. Mehta, J. Stander, and A. K. Jones. Application mapping onto a coarse-grained computational device. *submitted to European Journal of Operational Research*, 2007.

[8] J. E. Beasley. OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

[9] R. E. Bixby, S. Ceria, C. M. McZeal, and M. W. P. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, 58:12–15, 1998.

[10] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Chapman & Hall (Wadsworth, Inc.), New York, 1984.

[11] Melvin A Breuer. The application of integer programming in design automation. In *DAC '66: Proceedings of the SHARE design automation project*, pages 2.1–2.19, New York, NY, USA, 1966. ACM.

[12] David M. Cohen, Siddhartha R. Dalal, Michael L. Fredman, and Gardner C. Patton. The AETG system: An approach to testing based on combinatorial design. *IEEE Trans. Softw. Eng.*, 23(7):437–444, 1997.

[13] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1367–1372, October 2004.

[14] Reinhard Diestel. *Graph Theory*. Springer-Verlag: Heidelberg, 2005. 3rd edition.

[15] I. Sobel et G. Feldman. A 3x3 isotropic gradient operator for image processing. Never published but presented at a talk at the Stanford Artificial Project, 1968.

[16] R. Fisher, S. Perkins, A. Walker, and E. Wolfart. Image processing, 2004.

[17] C. H. Goldstick and D. G. Mackie. Design of computer circuits using linear programming techniques. *IRE-PGEC*, 11(4):518–530, August 1962.

[18] Joseph Haas, Maxim Peysakhov, and Spiros Mancoridis. GA-based parameter tuning for multi-agent systems. In Hans-Georg Beyer, Una-May O'Reilly, Dirk V. Arnold, Wolfgang Banzhaf, Christian Blum, Eric W. Bonabeau, Erick Cantu-Paz, Dipankar Dasgupta, Kalyanmoy Deb, James A. Foster, Edwin D. de Jong, Hod Lipson, Xavier Llora, Spiros Mancoridis, Martin Pelikan, Guenther R. Raidl, Terence Soule, Andy M. Tyrrell, Jean-Paul Watson, and Eckart Zitzler, editors, *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, volume 1, pages 1085–1086, Washington DC, USA, 25-29 June 2005. ACM Press.

[19] ILOG CPLEX. http://www.ilog.com/products/cplex/.

[20] Ming Jiang. Digital image processing, 2003.

[21] Thor Johnson, Neil Robertson, P. D. Seymour, and Robin Thomas. Directed tree-width. *Journal of Combinatorial Theory. Series B*, 82(1):138–154, 2001.

[22] Alex K. Jones, Raymond Hoare, Dara Kusic, Joshua Fazekas, and John Foster. An fpga-based vliw processor with custom hardware execution. In *FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, pages 107–117, New York, NY, USA, 2005. ACM.

[23] Ron Kohavi and George John. Automatic parameter selection by minimizing estimated error. In Armand Prieditis and Stuart Russell, editors, *Machine Learning: Proceedings of the Twelfth International Conference*, pages 304–312. Morgan Kaufmann, 1995.

[24] Evgeny B. Krissinel and Kim Henrick. Common subgraph isomorphism detection by backtracking search. *Software—Practice and Experience*, 34:591–607, 2004.

[25] Ratnesh Kumar, Wai Kit Leong, and Robert J. Heath. An integer programming approach to placement and routing in circuit layout, 2005. http://clue.eng.iastate.edu/ rkumar/PUBS/layout.ps.

[26] C. Lee, M. Potkonjak, and W. K. Magione-Smith. Mediabench: A tool for evaluating and synhesizing multimedia and communications systems. In *Proceedings of the International Symposium on Microarchitecture*, 1997.

[27] Young Hoon Lee, Kumar Bhaskaran, and Michael Pinedo. A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE Transactions*, 29(1):45–52, 1997.

[28] G. Mehta, C. J. Ihrig, and A. K. Jones. Reducing energy by exploring heterogeneity in a coarse-grain fabric. In *in Proceedings of the Reconfigurable Architecture Workshop*, 2008.

[29] Gayatri Mehta, Raymond R. Hoare, Justin Stander, and Alex K. Jones. Design space exploration for low-power reconfigurable fabrics. In *Proc. of the Reconfigurable Architectures Workshop (RAW)*, 2006.

[30] Gayatri Mehta, Justin Stander, Josh Lucas, Raymond R. Hoare, Brady Hunsaker, and Alex K. Jones. A low-energy reconfigurable fabric for the SuperCISC architecture. *Journal of Low Power Electronics*, 2(2), August 2006.

[31] Bruno T. Messmer and Horst Bunke. Efficient subgraph isomorphism detection: A decomposition approach. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):307–323, 2000.

[32] Zbigniew Michalewicz and David B. Fogel. *How to Solve It: Modern Heuristics*. Springer, December 2004.

[33] T.M. Mitchell. *Machine Learning*. McGraw Hill, Boston, 1997.

[34] George L. Nemhauser and Laurence A. Wolsey. *Integer and combinatorial optimization*. Wiley-Interscience, New York, NY, USA, 1988.

[35] S. Nissen. Implementation of a fast artificial neural network library (fann). Technical report, Department of Computer Science University of Copenhagen (DIKU), 2003. http://fann.sf.net.

[36] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2007. ISBN 3-900051-07-0.

[37] R. Niemann and P. Marwedel. Hardware/software partitioning using integer programming. In *In Proccedings of the European Design and Test Conference (ED & TC)*, pages 473–480, Paris, France, 1996. IEEE Computer Society Press (Los Alamitos, California).

[38] D. Rumelhart, G. Hinton, and R. Williams. Learning internal representations by error propagation. In D. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, pages 318–362. MIT Press, 1986.

[39] Justin Nathanial Stander. Electronic design automation for an energy-efficient coarse-grain reconfigureable fabric. Master's thesis, University of Pittsburgh, 2007.

[40] Selection Tool for Optimization Parameters (STOP). http://www.rosemaryroad.org/brady/software/.

[41] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976.

[42] Chiu wing Sham, Evangeline F. Y. Young, and Chris Chu. Optimal cell flipping in placement and floorplanning. In *DAC '06: Proceedings of the 43rd annual conference on Design automation*, pages 1109–1114, New York, NY, USA, 2006. ACM.

[43] C. F. Jeff Wu and Michael Hamada. *Experiments; Planning, Analysis, and Parameter Design Optimization*. John Wiley & Sons, New York, 2000.