

School Choice: A Discrete Optimization Approach

by

Justin W. Graham

B.S. Operations Research, U.S. Air Force Academy

Submitted to the Sloan School of Management
in partial fulfillment of the requirements for the degree of

Master of Science in Operations Research

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2020

© Justin W. Graham, 2020. All rights reserved.

The author hereby grants to MIT permission to reproduce and to distribute publicly paper and electronic copies of this thesis document in whole or in part in any medium now known or hereafter created.

Author
Sloan School of Management
May 1, 2020

Certified by
Dr. Dimitris Bertsimas
Boeing Professor of Operations Research
Thesis Supervisor

Accepted by
Dr. Georgia Perakis
William F. Pounds Professor of Management Science
Co-Director, Operations Research Center

THIS PAGE INTENTIONALLY LEFT BLANK

School Choice: A Discrete Optimization Approach

by

Justin W. Graham

Submitted to the Sloan School of Management
on May 1, 2020 in partial fulfillment of the
requirements for the degree of
Master of Science in Operations Research

Abstract

An equitable and flexible mechanism for assigning students to schools is a major concern for many school districts. The school a student attends dramatically impacts the quality of education, access to resources, family and neighborhood cohesion, and transportation costs. Facing this intricate optimization problem, school districts often utilize stable-matching techniques which only produce stable matchings that do not incorporate these different objectives; this can be expensive and inequitable. We present a new optimization model for the Stable Matching (SM) school choice problem which relies on an algorithm we call Price-Costs-Flexibility-and-Fairness (PCF²). Our model leverages techniques to balance competing objectives using mixed-integer optimization methods. We explore the trade-offs between stability, costs, and preferences and show that, surprisingly, there are stable solutions that decrease transportation costs by 8-17% over the Gale-Shapley solution.

Technical Supervisor: Dr. Dimitris Bertsimas
Boeing Professor of Operations Research

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgements

There are many people along this incredible journey at MIT who have made this season of life remarkable.

First and foremost, I want to thank Dimitris. From the start of our time together, his dependable optimism, extraordinary intellect, and steadfast ability for hard work has never wavered. I have come across none better than Dimitris in identifying a problem, and creating a solution that truly changes the operations research world. He teaches mathematical concepts to me, but more importantly he teaches me what it really means to be committed towards a meaningful cause, and how to care for the well-being of others.

Thank you to Jean, Arthur, Ted, Scott and all the members of the ORC for helping me along the way and creating a fun environment to work.

Thank you to my parents, Tim and Kellie. I won the parent lottery and could not be more thankful for the love and support. My time in Cambridge has been a tremendous period of humbling growth and breakthrough, and without them, none of it would have been possible. And finally, without my roommates Charlie and Jackson, graduate school would not have been the legendary experience it was. Thanks for making this home.

Contents

1	Introduction	8
1.1	Literature Review	9
1.2	Contributions	10
1.3	Structure	10
2	A Binary Optimization Model	12
2.1	Solution Methodology	13
2.1.1	Generating Constraints	14
2.1.2	Randomized Rounding	14
2.2	Trade-Off Assessments	15
2.2.1	Parameter Search Methodology	15
2.3	Results	16
2.3.1	Data	16
2.3.2	Complexity	17
2.3.3	Application to BPS	17
3	Parallelizing Large-Scale Optimization	20
3.1	Optimizing Serial Code	21
3.1.1	Memory	21
3.1.2	Avoiding Heap Allocations	21
3.1.3	Julia’s Type Inference and Function Specialization	22
3.1.4	Results of Serial Code Optimizations	22
3.2	Network Analysis	23
3.2.1	Amdahl’s Law	24
3.2.2	Communication Network	24
3.2.3	Coordination Scheme	25
3.3	Parallel Implementation	26
4	Conclusion	28
4.1	Future Research	28
A	BPS Budget	30
B	Additional Results	31

List of Figures

2.1	Randomized Rounding Geometry	15
2.2	BPS Demographics	16
2.3	BPS Complexity	17
2.4	Efficient Frontier Trade-Offs	19
3.1	Memory Hierarchy in Modern Computers	21
3.2	Flame Graph Profiling	23
3.3	Latency, Bandwidth, Inverse Bandwidth Plots	25
3.4	Node Workloads	26
A.1	Share of BPS budget going to transportation costs [2].	30
B.1	Trade-off between school and family preferences.	31

List of Tables

2.1	Maximizing Family Preferences	17
2.2	Maximizing School Priorities	18
3.1	Naive vs. Optimized Code Formulations	22
3.2	Execution Time Comparisons	27

Chapter 1

Introduction

From educating an increasingly diverse student body to adapting to technological change, school districts across the country are confronted with challenges on a daily basis. Among the most difficult challenges, perhaps, is an assignment system of students to schools. School choice problems are some of the most widely discussed topics in the educational field. President Trump deems it an issue of importance, calling to extend equitable school choice mechanisms to millions more children across the US [28]. On a more local level in Boston, officials have considered this problem closely since the mid-90's, adopting a deferred acceptance algorithm in 2003, and eventually iterating to the currently used HomeBased Assignment Plan in 2013. Advocates of a school choice mechanism - over the commonplace neighborhood districting system - also argue that it is another way to inject competition into the public school marketplace by encouraging competitive pressure. However, it is a very difficult problem given the social and political challenges of creating an equitable assignment mechanism. Given the unquestionable racial and economic impact school choice has, it is clearly a politically charged problem. The process must prioritize the preferences of both students and schools, while simultaneously ensuring equal access to quality schools across all demographics. Implementation of the most cost effective solutions present a challenge, especially when the mechanism requires students to change schools from year to year.

In a classical school choice problem, there are students who need to be assigned a seat at a school. Schools have maximum capacities, as well as a set of diversity quotas that must be met. Each student has preferences over schools, and each school has priorities for different students. Traditionally, students are assigned to schools according to where they live. However, in an effort to give families more input in where their children attend school, deferred-acceptance (DA) mechanisms based on preferences are used. Usually, the current solution to the school choice problem is some variation of the Gale-Shapley [13] matching algorithm (a version of DA), which has a number of desirable outcomes:

1. **Stability** - no student-school pair (i, j) exists such that student i prefers school j to their current assignment and student i has a higher priority at school j . This reflects a notion of procedural fairness in how pairs are formed, as well as outcome fairness because it always yields matchings that are in the best interest of the students.
2. **Incentive Compatibility** - Students have no motive to try and manipulate the assignment

system because truthful preference revelation is the dominant and preferred strategy.

Despite these properties, it is far from obvious that the Gale-Shapley algorithm effectively captures and implements the priorities of school districts. Complaints of a stable-matching system include a high transportation burden and low community cohesion, among others. For example, Boston Public Schools (BPS) increased their transportation spending to \$123 million in 2018, over 10% of the entire budget. BPS transportation costs are the second highest in the nation, more than five times larger than the average of the largest 200 school districts [20]. It is clear that school districts must improve their mechanisms in order to control budget concerns. The goal of this work is to improve upon school choice matching algorithms through techniques of modern linear and robust optimization.

1.1 Literature Review

The two-sided stable matching (SM) problem arises when there are a set of i students $\{p_1, \dots, p_n\}$ and m schools $\{s_1, \dots, s_m\}$ with each having an ordered list of preferences for their preferred matching. The stable matching problem asks for a matching of the students and schools that exhibits the following stability property: there does not exist a student i and school m who are not matched under the current assignment, but prefer each other to their current assignment. These pairings are known as blocking pairs.

Generalizations of the SM problem exist in which the preference lists of the students and schools are allowed to contain ties, and the preference lists are bounded. Such restrictions arise naturally in practical applications such as school choice, hospital-doctor matching, and kidney donations [3]. When complete enumeration of all student preferences list without ties is unavailable, the problem is called stable matching with ties and incompleteness (SMTI). A stable matching always exists for SMTI and can be easily obtained by arbitrarily breaking ties, but the stable matching for one instance may have a different number of matches than another. More specifically applied to our context, one solution could result in more students receiving their preferences than another solution that is still stable in the solution space, but not optimal in the sense that the most students are awarded the choice they prefer. In this context, various definitions of stability exist, namely weak, strong, and super stability. Weak stability is our original definition of stability, occurring when no blocking pairs exist. This will be referred to as simply "stability" moving forward, as it is the strictest and most appropriate definition.

The way in which ties are broken yield matchings of various cardinalities. It is desirable to find the maximum SM (MAX SMTI) where the maximum amount of families are awarded their preferences, but this is well known to be an NP-hard problem [14].

There is existing literature on heuristics to solve these types of problems. State-of-the-art methods have been shown to achieve a best known performance guarantee approximation of $3/2$ with respect to the matching objective. Delorme et. al. [11] reduce computation time to under one hour by using preprocessing techniques and introducing dummy variables to reduce the number of non-zero entries.

Adaptive Search (AS) methods have become the most used meta-heuristic for solving larger instances of SMTI. AS takes advantage of the model in terms of constraints and variables in

order to guide the search more precisely than a traditional cost function. AS starts with a random solution configuration, and iteratively improves it one variable at a time. Podhradsky perform a thorough experimental study of eight AS algorithms on data sets with sizes up to 733 students and 43 schools [22].

In their Nobel prize work, Gale & Shapley [13] prove the existence of a stable marriage arrangement for any pattern of preferences. In research more specific to our work, Abdulkadiroglu & Sonmez [5] offer two solutions for the school choice problem: Gale-Shapley matching and a top trading cycles mechanism. Later, Abdulkadiroglu [6] realizes the shortcomings of these deferred acceptance mechanisms and expands upon his work by allowing for students to communicate their preference intensities to influence how they are treated in ties. Within a set of Gale-Shapley assignment outcomes, Bodo-Creed [10] evaluates the tension between student welfare, encouraging neighborhood schools, and diversity within schools. Lastly, Shi [27] uses correlation optimization techniques to improve neighbors’ chances of going to the same school from the output of a deferred-acceptance algorithm.

However, these improvements to school choice models do not incorporate transportation costs when assigning students. Belford & Ratliff [7] are the first to offer a theoretical procedure for assigning students that takes cost into account in 1971 through a minimum-cost network flow formulation. [30] proposes one of the first linear optimization formulations for a stable-matching problem, but without regard to cost or fairness. One reason for relatively sparse literature in the area is due to the difficulty in computing a stable matching problem to optimality. Optimization solvers are not able to solve real-world problems even in hours. Delorme et. al. apply pre-processing techniques and alternative stability constraint formulations as heuristics for the stable matching problem, and achieve fast formulations at the expense of losing provable optimality on a medium-sized problem.

1.2 Contributions

1. We formulate the multi-objective problem of balancing stability, preferences and (racial, gender and income) diversity as a binary optimization model.
2. We present a computationally tractable method to solve large scale matching linear optimization problems.
3. We offer a framework for decision makers to identify the range of achievable outcomes and the inherent trade-offs between competing objectives.
4. We implement a supervisor-worker parallel coordination scheme to speedup the time for querying multiple instances of the school choice model.
5. Applied to Boston Public Schools, we show a decrease in transportation costs by 8-17% over the Gale-Shapley solution, significant savings for the school district.

1.3 Structure

The structure of the paper is as follows:

1. In Chapter 2, we introduce an integer optimization approach to solving a version of SMTI. The objective function and constraints of the model are discussed in detail. Section 2.1 describes our methodology for computing solutions tractably, while Section 2.2 details multi-objective trade-off assessments. From these developments, we present results applied to a real-world school choice problem with BPS in Section 2.3.
2. Such a large problem instance like the model presented in Chapter 2 is difficult to tractably solve. In Chapter 3, we implement a supervisor-worker parallel coordination scheme after taking careful consideration for important metrics such as Amdahl's law. A speedup by over a factor of three is achieved by parallelizing the optimization algorithm.
3. In Chapter 4, we conclude and propose future avenues for research.

Chapter 2

A Binary Optimization Model

In this chapter, we formulate a binary optimization problem that assigns students to schools that balances preferences, costs as well as achieving racial, gender and income diversity. Let $[p] = \{1, 2, \dots, p\}$ the set of schools and $[n] = \{1, 2, \dots, n\}$ the set of students. We introduce the following data the problem uses:

- (a) Each student i has a ranked list of schools, denoted by r_{ij} the rank of school j by student i . We let F_i be the set of schools included in student i 's rankings. Student i can only be assigned to one of these schools in the set F_i . We use the notation $r_{ij} < r_{ik}$ to denote that student i prefers school j over school k .
- (b) School priorities are determined by two factors: the proximity of a student to a school (with the closest distance taking priority), and family considerations (i.e., siblings should be given priority for the same school). We let q_{ij} the priority of student i by school j (smaller values correspond to higher priority). We let E_j is the set of students eligible to be assigned to school j . We use the notation $q_{ij} < q_{kj}$ to denote that school j prefers student i over student k .
- (c) Each school j has capacity C_j , i.e., school j can be assigned to at most C_j students.
- (d) We define the cost of assigning student i to school j as c_{ij} . For simplicity, we use the Euclidean distance as a proxy for the transportation cost for student i to school j . Transportation costs are normalized onto the same scale as the preference vectors for fairness and equal weighting in the objective function.
- (e) We let D_k , $k = 1, \dots, 6$ the set of students of race k with $k = 1$: Hispanic, $k = 2$: Black, $k = 3$: White, $k = 4$: Asian, $k = 5$: male, and $k = 6$ low-income students.
- (f) We let L_{jk} be the absolute number of students in school j of race k that violates the minimum race requirement. U_{jk} is likewise defined for the maximum diversity percentage.
- (g) As the problem is multi-objective, we use parameters λ_n , and $n = 1, \dots, 5$ to control the significance of cost, student and school preferences, and diversity goals.

We define the following decision variables:

$$x_{ij} = \begin{cases} 1, & \text{if student } i \text{ is matched to school } j \\ 0, & \text{otherwise} \end{cases}$$

The problem of optimizing cost and preferences is formulated next. We use the name Preferences-Costs-Flexibility-and-Fairness Model PCF²:

$$\min \quad \sum_{i \in [n]} \sum_{j \in F_i} \sum_{k \in D_{[4]}} \lambda_1 c_{ij} x_{ij} + \lambda_2 r_{ij} x_{ij} + \lambda_3 q_{ij} x_{ij} + \lambda_4 L_{jk} + \lambda_5 U_{jk} \quad (2.1)$$

$$\text{s.t.} \quad \sum_{j \in F_i} x_{ij} = 1, \quad \forall i \in [n], \quad (2.2)$$

$$\sum_{i \in [n]} x_{ij} \leq C_j, \quad \forall j \in [p], \quad (2.3)$$

$$x_{ij} + \sum_{k: r_{ij} < r_{ik}} x_{ik} + \sum_{k: q_{ij} < q_{kj}} x_{kj} \leq 1, \quad \forall i \in [n], \quad \forall j \in F_i, \quad (2.4)$$

$$\alpha_k \sum_{i \in [n]} x_{ij} - \sum_{i \in D_k} x_{ij} \leq L_{jk} \quad \forall j \in [p], \quad \forall k \in [6], \quad (2.5)$$

$$\sum_{i \in D_k} x_{ij} - \beta_k \sum_{i \in [n]} x_{ij} \leq U_{jk} \quad \forall j \in [p], \quad \forall k \in [6], \quad (2.6)$$

$$x_{ij} \in \{0, 1\}.$$

The objective (2.1) balances cost, student preferences, school preferences, and diversity objectives. Constraint (2.2) ensures that each student is assigned to exactly one school while Constraint (2.3) ensures school capacity limits are not exceeded. Constraint (2.4) ensures stability of the assignment. We note that the first summation means student i preferring school j over school k , while the second summation represents school j preferring student i to student k . If $x_{ij} = 0$, then not both $\sum_{k: r_{ij} < r_{ik}} x_{ik}$ and $\sum_{k: q_{ij} < q_{kj}} x_{kj}$ can be equal to one, since in this case student i and school j are matched to less favorable pairings than each other, and therefore the matching is not stable. Moreover, if $x_{ij} = 1$, then both sums are zero. In both cases, the constraint is a valid formulation for the assignment to be stable. The particular constraint for ensuring stability is from [29] and [12]. Constraints (2.5) and (2.6) place lower and upper bounds on all the diversity sets, respectively. School metrics on the proportion of African-American, White, Asian, Hispanic, gender, and low-income students must fall within a certain range in an effort to maintain a sense of social fairness. These bounds can be adjusted based on the goals of the school district.

We note that Problem (2.1) is integral, see [29] and [12].

2.1 Solution Methodology

Formulation (2.1) has np binary variables, $O(np)$ constraints. Given that our target is to solve problems with $n = 22,420$ and $p = 129$, the number of students and schools for Boston Public Schools, formulation (2.1) is not directly solvable by state of the art integer optimization solvers. To overcome this, we first define x_{ij} only for those pairs that are feasible for both students and schools to encode sparse decision variables.

2.1.1 Generating Constraints

The classical large-scale optimization cutting planes, or delayed constraint generation method is critical for finding optimal solutions in a tractable manner. The key bottleneck of the problem is the large number of stability constraints (2.4) of which only a small subset is binding. Cutting planes methods allow us to solve (2.1) by solving a sequence of linear optimization problems.

Instead of dealing with all of the stability constraints, we consider a subset I , relax the integrality requirements, and form the relaxed problem with these conditions. Initially, we begin with zero constraints in the subset I and find an optimal feasible solution x^* to the relaxed problem. Two possibilities arise:

1. Suppose x^* is feasible for the full original problem, then x^* is optimal and we terminate the algorithm.
2. If x^* is infeasible for the original problem, we find a violated constraint, add it in to the set I , and continue.

2.1.2 Randomized Rounding

Our relaxation means that some x_{ij} decision variables in the optimal solution are fractional. However, we use randomized rounding methods in our algorithm to guarantee an integral and optimal solution [12]. Let F be in set of stable matchings, i.e. all integer vectors satisfying PCF², and let P_{SM} be in the polyhedron describing the linear relaxation for our formulation. We can show that $P_{SM} = \text{conv}(F)$. Under the proposition that \mathbf{x} is a feasible solution to the polyhedron, then constraint (2.4)

$$x_{ij} + \sum_{k: s_k < p_i s_j} x_{ik} + \sum_{k: p_k < s_j p_i} x_{kj} = 1$$

The proof can be found in [12], Chapter 4. From this proposition and using the geometry of the solutions to P_{SM} seen in Figure 2.1, we apply the following randomized rounding algorithm to express a fractional solution as a convex combination of integral stable matchings.

1. Generate a random number U uniformly over $[0,1]$.
2. Construct a matching in the following way: Match student i to school j if $x_{ij} > 0$ and in the row corresponding to student i , and U lies in the interval spanned by x_{ij} in $[0,1]$. Match school j to student i if in the row corresponding to school j , and U lies in the interval spanned by x_{ij} in $[0,1]$.

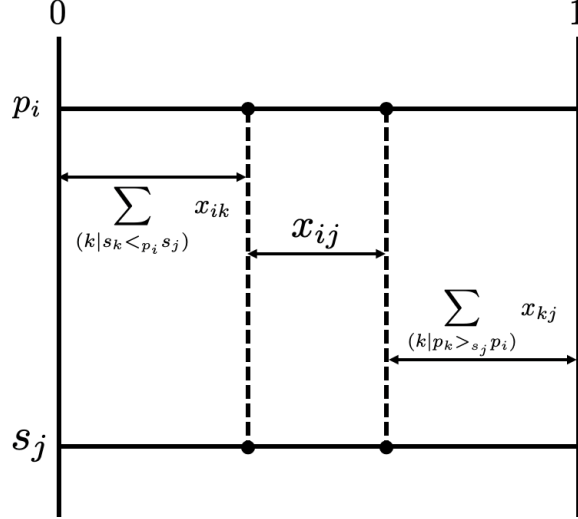


Figure 2.1: Geometry of the solutions to a randomized rounding formulation.

In this way, student i is matched to school j if and only if school j is matched to student i . Furthermore, there are no double matchings, so constraint (2.2) is not violated. Hence, it gives rise to a perfect stable matching.

2.2 Trade-Off Assessments

Even with state-of-the-art solvers and computation power, one instance of the Price-Costs-Flexibility-Fairness algorithm under full stable matching conditions requires nearly 24 hours to solve. However, we need qualitative assessments of trade-offs in multiple objectives (cost, family preferences, school preferences, and diversity quotas) to fully explore the solution space, which necessitates an understanding of the full range of achievable outcomes. This is especially challenging in the case where the decision maker does not coincide with the optimization expert, as is the case with our school choice problem. In this section, we address this challenge by introducing a setting where BPS decision makers can work directly and intuitively from a set of desirable outcomes, back to the design decisions that achieve them. Importantly, our framework is designed to allow for quick iteration as the decision maker's understanding of what is achievable and the inherent trade-offs therein changes.

2.2.1 Parameter Search Methodology

Our PCF² model contains parameters that weigh multiple objectives. A different set of parameters leads to different objective outcomes, and the "optimal" parameters are difficult to find. Querying the PCF² model iteratively is computationally expensive, so rather than attempting to tune parameters through a grid search approach, we move directly from objective outcomes back to the parameters in a lookup-table fashion. This framework offers a tool for efficient iteration and refinement of proposed policies by working directly with their desired outcomes.

First, since the domain of our BPS data is in the form of a four dimensional hyper-rectangle ($\lambda_1, \lambda_2, \lambda_3$ and λ_4 with $\lambda_5 = 0$), we use latin-hypercube sampling as an efficient method for

generating representative design points, more diverse than uniform sampling when N is relatively small. This ensures our parameter points are not all sampled from too small of a space. We use many nodes on MIT Supercloud’s TX-GAIA Supercomputer, to query PCF² $N \approx 1000$ samples.

This produces an set of achievable outcomes in the form of an efficient frontier (visualizations found in Section 2.3). The key idea is that as our understanding of trade-offs and the range of achievable outcomes develops, we provide solutions anywhere along the efficient frontier of outcomes.

2.3 Results

The overall goal of the work is to reduce transportation costs via the student assignment mechanism in an effort to save BPS money which they could reinvest back into the students. The results are encouraging: 8% – 17% savings in cost.

2.3.1 Data

We use a combination of data supplied by BPS and simulated data to apply our problem formulation. Student location, school locations and the student’s current school assignment are available through BPS. Race and incomes are generated according to the BPS Demographic data from the table below [2]. For preference rankings, families are given the option to rank up to 20 schools on the menu of options, and these preferences are generated in such a way that a student chooses a nearby school 65% of the time and a random school the other 35% of the time. School preferences for students are solely determined by the proximity of a student to a school, with the closest distance having highest priority.

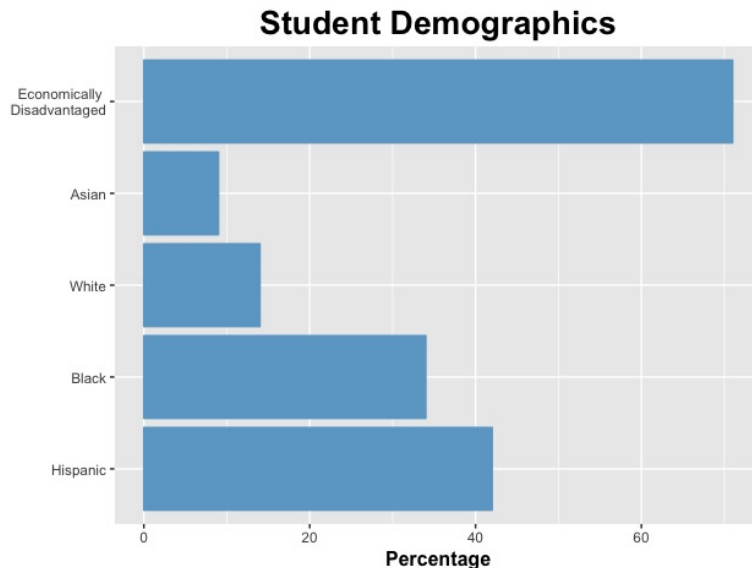


Figure 2.2

2.3.2 Complexity

The BPS problem is a very large instance of the SMTI problem. There are 22,420 students requiring matchings to 129 schools across the district. Each family is allowed to rank up to 20 schools, with seven of them required to be nearby. The problem calls for over 3.3 million integer decision variables, 1,290 capacity and diversity constraints, and 448,400 stability constraints. Our sparsity pre-processing reduces the number of decision variables by a factor of 10 down to 448,400. Still, the current state-of-the-art literature has experienced difficulty solving problems of this scale to optimality, but our methods allow us to accomplish the task.

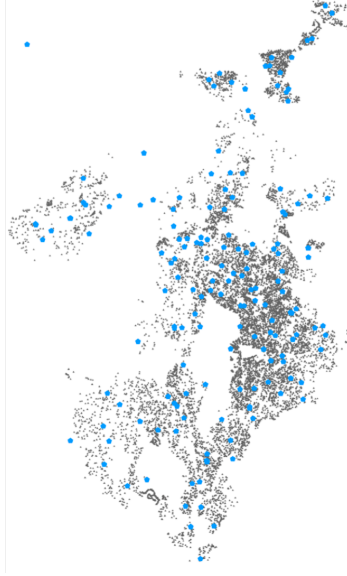


Figure 2.3: Map showing the BPS school district with blue circles representing schools, and black dots representing student locations. The complexity of the problem is large [9].

2.3.3 Application to BPS

Table 2.1 shows the comparisons between model formulations when family preferences are maximized. The Gale-Shapley assignment model is the option currently used by many school districts, where transportation costs are not a factor in the model. In our PCF² formulation, we see an 8.0% decrease in the cost with a negligible loss in family preferences. We represent the cost of student-school assignments with a proxy for the transportation cost. Here, we use the city-block distance metric as a cost estimate.

	Current BPS	PCF ² Algorithm
Cost Improvement	-	8.0%
Family Top 3 Choices	87.7	86.2
School Top 3 Choices	71.0	72.8

Table 2.1

	Current BPS	PCF ² Algorithm
Cost Improvement	-	17.2%
Family Top 3 Choices	57.2	53.1
School Top 3 Choices	98.1	98.1

Table 2.2

Table 2.2 shows a similar comparison between model formulations, but this time we optimize the model to maximize school priorities. In this setting, we see a 17.0% decrease in the total cost, once again with a negligible loss in family preferences.

Next, the price of awarding families their top preferences is illustrated. All of the points in the charts below are stable matchings, and the red point represents the cost and percentage of families who receive their top three school listings under BPS’s current Gale-Shapley matching method. In the top figure, as the points become a lighter shade of blue, the algorithm places more emphasis on reducing cost rather than preferences. It is clear that as more families are given their top choices, the cost of the solution also increases to form a clean efficient frontier. A major benefit of the PCF² model is the ability to offer multiple solutions. Boston Public School’s officials can determine a cost level and preference level, and this figure depicts if the solution is feasible and optimal. The bottom figure represents the same trade-off, but with school priorities.

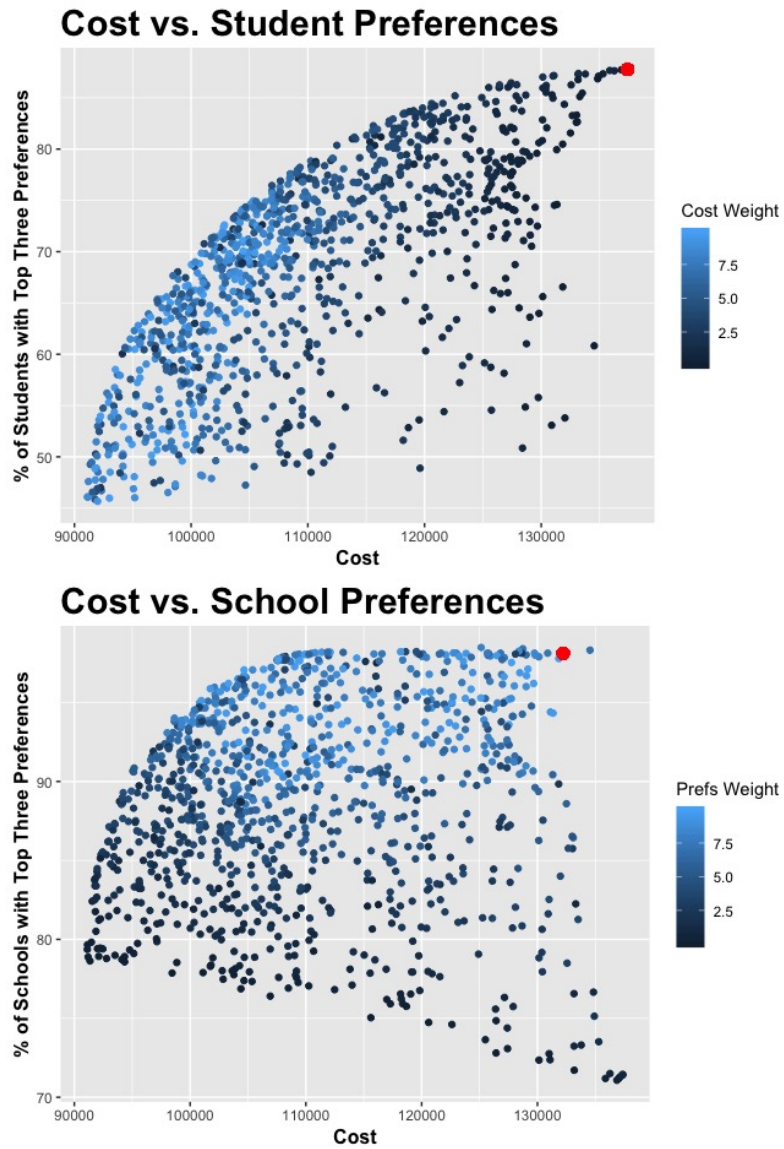


Figure 2.4: Top: Trade-off between cost and family preferences. Bottom: Trade-off between cost and school preferences. The figures also show the cost savings over stable solutions that don't account for cost.

Chapter 3

Parallelizing Large-Scale Optimization

In order to achieve the results in Chapter 2, we must employ parallelization strategies to solve for solutions quickly. This chapter details our method of parallelizing the PCF² model.

Mixed-integer and discrete optimization (MIP) methods at its core is finding the root of a linear function over linear constraints subject to restrictions on some of the variables in the model. Sequential algorithms for MIPs have improved substantially in the last decade and are now efficiently solved by state of the art solvers such as Gurobi. Parallel processes for these types of problems have not seen the same level of improvement, however [24]. In some ways, progress on sequential solutions has been detrimental to parallelization ease and efforts.

Solutions to these classes of problems represent a complete enumeration of the solution space in the form of a branching tree. A branch and bound (BnB) algorithm very naturally lends itself to be implemented on multiple processes in parallel since we may distribute sub-trees to different compute cores [24]. The partitioning into smaller sub-problems narrows the feasible region until the optimal solution is found. The power of BnB comes from the bounds used to truncate the search.

The literature on parallelizing BnB methods is robust. Work first began in the 1970's, and better performance techniques have continued to be introduced. As an example of some recent work, Bergman et. al. address the problem of search schemes during the sub-problem phase [8]. BnB relies heavily on the upper and lower bounds of the search, which presents a challenge when solving many sub-problems in parallel. Either the workers must communicate often, or much of the work will be a redundant effort. Bergman uses decision diagrams in order to approximate bounds of tree sub-problems to make parallelization more efficient. Linderoth [17] proposes a parallel algorithm that focuses on generating cutting planes near the top of the BnB tree to flexibly solve MIPs. He uses the algorithm to solve a large instance of the Set Partitioning Problem in a reasonable amount of computing time.

The limits to scalability for solving MIPs are well understood, and much has been overcome by the existing approaches in the literature. For this reason, we will use Gurobi's excellent automatic internal BnB parallelization scheme and focus our efforts on a coarser parallelization strategy.

3.1 Optimizing Serial Code

Before this program is run in parallel, it is important to understand that at the center of any fast parallel code is fast serial code. Since parallelism is a performance multiplier, beginning from a poor position is a recipe for slow and unexpected performance. It is addressed in our work by improvements in introducing non-allocating functions, and taking advantage of Julia’s type-stability features. For the remainder of this section, performance increases are measured against a naive working example with no optimizations. First though, to start optimizing code, it is important to understand the memory model of your computer.

3.1.1 Memory

Modern computers have multiple CPU’s, and each CPU has multiple cores. At a high level, a CPU’s core memory accesses different levels of cache memory as seen in Figure 3.1. Optimized code has the ability to use things in a closer cache so data is queried less often.

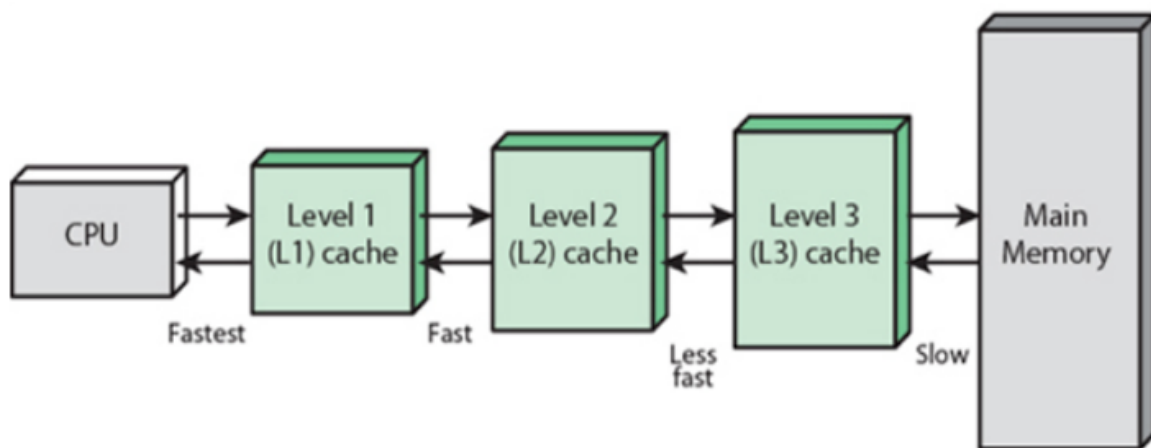


Figure 3.1: Memory hierarchy in modern computers.

Moving down to a lower-level view of memory, each box in Figure 3.1 is composed of a stack and a heap. The stack is ordered and is accessed very fast, but the size of variables must be known at compile time. The heap exists because this is not always possible, so it is composed a a series of pointers to data in main memory, but it is very costly as it requires allocations to pull data up the cache chain to be accessed.

3.1.2 Avoiding Heap Allocations

Heap allocations must locate and prepare a space in memory, so we want to avoid heap allocations if at all possible. Several times in our PCF² Algorithm, information must be written into a data structure (either an array or `JuMP.Containers.SparseAxisArray`, JuMP’s version of a two dimensional sparse array). Usually writing into an array involves heap allocations, but if mutation is used (changing the values of an already existing array), then no heap allocations occur. In our program, instead of creating an array each time, we utilize a cached array.

Slicing in Julia also produces copies of the data structure, thus allocating to a new output. The `@view` command avoids heap allocating by asking for a view of the data rather than a copy.

3.1.3 Julia’s Type Inference and Function Specialization

Julia is the ideal programming language because of core design features such as multiple dispatch, type inference and function specializations.

When the compiler executes a computation, it will ask for the type of the object in order to know how to do the computation correctly. When the type is not known, there is overhead run-time and memory costs as the variables could be changing in size and value, causing a heap-allocation. Julia, however, runs a type inference algorithm to discover the types of objects just before compilation, making the code fast.

Much of the speed improvement in our program can be pointed towards type specialization in functions. Even if the type of a function is not manifested at compile time, Julia’s **generic function** feature interprets a function over all possible methods, where each method’s type *is* known. To take advantage of this, we split our single naive function into multiple functions that become inputs to each other. In this way, type specialization speeds up code and reduces allocations.

Finally, multiple dispatch in Julia allows the programmer to tell functions how to compute based on different type assertions on the input values. Our functions all have the most strict version of the type defined on the inputs. This is what makes code much faster than our naive implementation (or anything in Python): the program is specializing on its types on each function, and those functions infer type outputs, which can be used by the compiler to generate the most efficient function [23].

Automatic bounds checking is disabled as one final small code optimization. All of these features are covered in depth in the Julia documentation [1].

3.1.4 Results of Serial Code Optimizations

After all of these program modifications, our PCF² program exhibits a drastic performance increase in serial. These results are run on only a subset of the BPS data set for the sake of speed, but extrapolate according to the complexity schemes from Chapter 1. The original naive implementation—how a MIP formulation would normally be coded—is described in Table 3.1 in relation to new MIP codes. We see 35% fewer allocations, 41% less memory used, and speedup by a factor of 2.5.

	Naive MIP Code	One-time Model Build	Optimized Code	Total Optimized	Speedup Imp
Allocations	67,849,436	9,699,783	34,746,147	44,445,930	34.5%
Memory (GB)	2.330	0.737	0.641	1.378	40.9%
Mean Runtime (s)	47.86	6.34	12.96	19.30	2.48 x

Table 3.1: Comparison of naive vs. optimized model formulations. There is a dramatic performance increase from implementing serial code optimizations like non-allocating functions, function specialization, and multiple dispatch.

Profiling the code through `Profile.jl` reveals the bottlenecks of the program. Approximately 97% of the clock-cycles are due to building the optimization model in JuMP. An intuitive visualization of the run-time is a flame graph as seen in Figure 3.2. The horizontal length represents the amount of time spent on that function. Because the main driver of the run-time involves model building, the new code splits the algorithm into two functions: one to build the

model and another to optimize it. This structure means there is only a one-time cost involved to building the model where the majority of time is spent, and the parallel portion of the algorithm will not need to invoke the build for every solve.



Figure 3.2: Flame graph produced through `Profile.jl`. The graph is a visual depiction of the number of clock-cycles per process in the program. The most time consuming computations all deal with building the model: creating variables, and adding them into the model.

3.2 Network Analysis

The challenge of parallel MIP algorithm architectural design comes primarily from balancing the workload across the nodes, and limiting the amount of communications between nodes. A complete parallel algorithm can be viewed as a collection of mechanisms that can be split into two groups: governors of the parallelization strategy (i.e. data movement) and programs that determine what each worker should be doing [19]. The core issue that must be addressed is to how to handle the globally useful information for the algorithm I/O. We want to move as little information between processors as possible, but enough to avoid losing valuable computational efficiency.

A parallel programming model (PPM) describes how the software is going to implement the the program in parallel. Three common PPM's are used: threading, message passing, and global arrays [16]. We will focus on the messaging PPM. This type of model requires that all processors be able to send and receive messages to each other. In the computing community, Message Passing Interface (MPI) is the standard infrastructure, so we implement `MPI.jl` in our work. MPI is composed of an interface standard and a set of associated libraries for allowing separate processes running on separate cores to communicate with each other via shared memory or over a communication network [26].

As a note: in this shared-memory architecture, the issue of memory contention and lock schemes present themselves, but we do not address these ideas, instead relying on `MPI.jl`'s internal implementations.

3.2.1 Amdahl's Law

An important parallelization measure is speedup. The speedup measures how much faster the parallel algorithm performs relative to the sequential algorithm. One of the most important design concepts in parallel implementations in managing the overhead introduced by a message passing network. The total work that a program must compute can be broken into a part that can be done in parallel, and a part that must be done on one processor (the supervisor) [16].

$$W_{\text{total}} = W_{\parallel} + W_{\text{serial}}$$

Assuming this equation, then the execution time scales with $T(N_P) = \frac{W_{\parallel}}{N_P} + W_{\text{serial}}$ which translates to a speedup of

$$S(N_P) = \frac{W_{\text{total}}}{\frac{W_{\parallel}}{N_P} + W_{\text{serial}}}$$

In the theoretical case when the number of processors is very large, the maximum possible speedup of a single program as a result of parallelization, known as Amdahl's Law, is

$$S_{\text{max}} = \frac{W_{\text{total}}}{W_{\text{serial}}}$$

3.2.2 Communication Network

The underlying communication network in a parallel model has several important characteristics: namely latency and bandwidth.

Latency is a measure of how long it takes for data to travel between processors in the network. Bandwidth, typically measured in bytes per second, is the maximum rate at which data can flow over a network [16]. The computing cluster we use is the MIT Lincoln Laboratory's TX-GAIA Supercomputer [4]. It is composed of 52 nodes with 1348 cores and 873 TB of storage space. The charts below show the latency and bandwidth curves for the Intel Xeon E5-2650 cores that we use for parallelizing the MIP. The inverse bandwidth plot is even more helpful to provide a guide on how many operations that need to be performed on an object to amortize the cost of communicating that value [16]. To calculate this, the processor speed (measured in floating point operations per second FLOPS) is divided by the bandwidth to yield the number of FLOPS that can be performed in the time it takes to send a message of a given size.

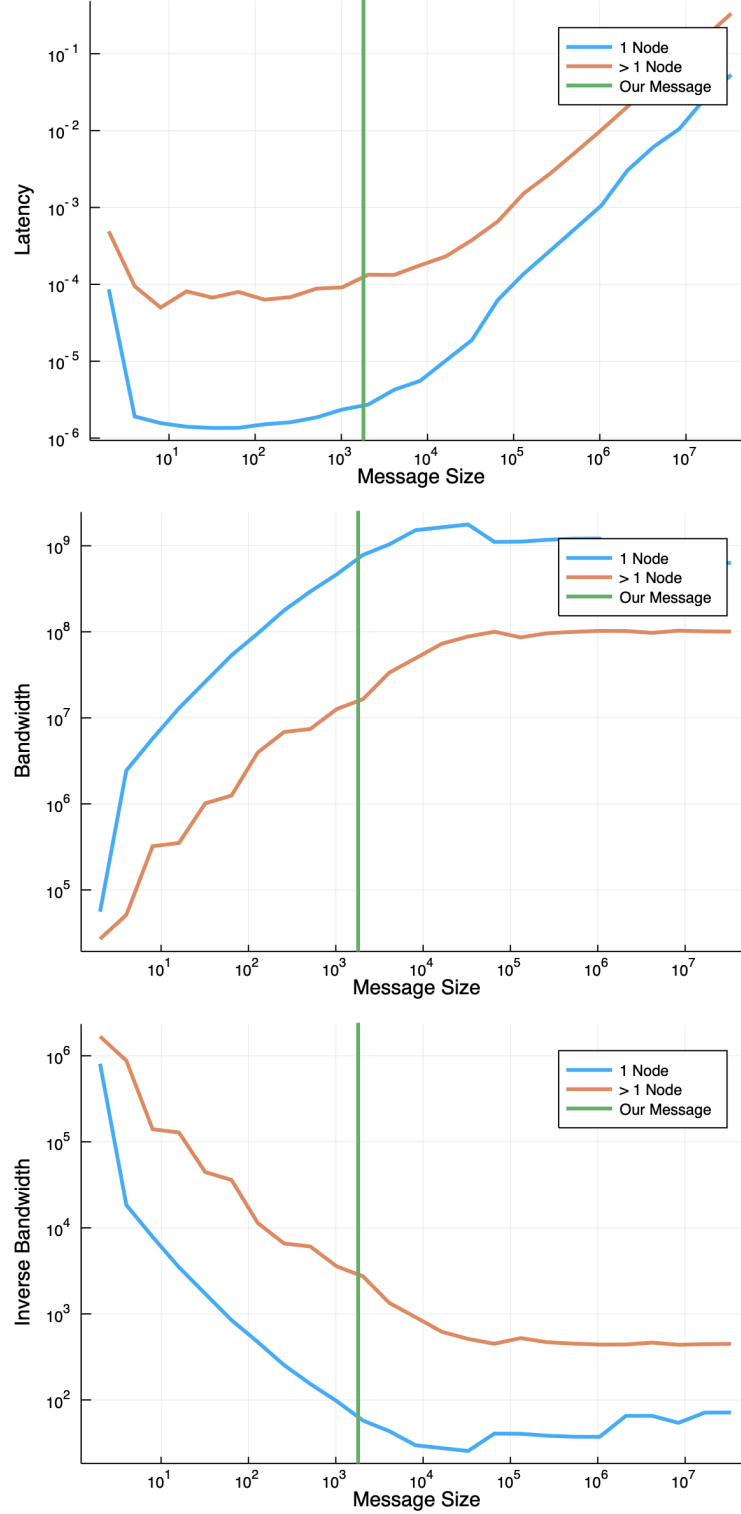


Figure 3.3: Latency, bandwidth, and inverse bandwidth as a function of message size for the MIT Supercloud High Performance Computing system.

3.2.3 Coordination Scheme

We choose a supervisor-worker coordination scheme to orchestrate the parallelism. The idea of this model is that the supervisor processor's function is to coordinate the workload through a distributed load balancing scheme. An important decision in the parallelization strategy is the

distribution of a set of programs that can be solved in parallel. In our application, we know a priori the difficulty of each problem being distributed, allowing us to easily balance the loads to each of the processes so that they are doing similar amounts of work in a round-robin fashion.

Our computation will be distributing the same program with different parameter values to each worker. Therefore, it can be divided into independent parts, executed by separate processes that requires a very small amount of communication. This architecture is known as an embarrassingly parallel computation and requires almost no interaction between workers [31].

3.3 Parallel Implementation

After all the above discussion, it is time to test the results of how our program runs in parallel. The first metric to compute is the overhead cost of the additional MPI code and executing the program on the Supercloud computer. The time to compute one instance of the serial MIP is 25.59 seconds, while the time to execute the same exact computation, but this time including the MPI overhead, is 44.82 seconds. This indicates that the cost of parallelization is very expensive in our case.

Each process (worker) in the network is computing the one-time model build, then optimizing the MIP for each of the parameter values it is distributed by the round-robin technique. In total, we are solving for eight parameter values (i.e. optimizing the model eight times). Table 3.2 shows the scaling analysis for various number of workers and nodes. Note that when parallelizing a MIP, Gurobi requires two full cores to operate regardless of the problem size. Figure 3.4 provides a visualization of how the program is running on multiple nodes. The coordination scheme is working nicely, as all the computations are occurring on nodes physically close to each other on the network.

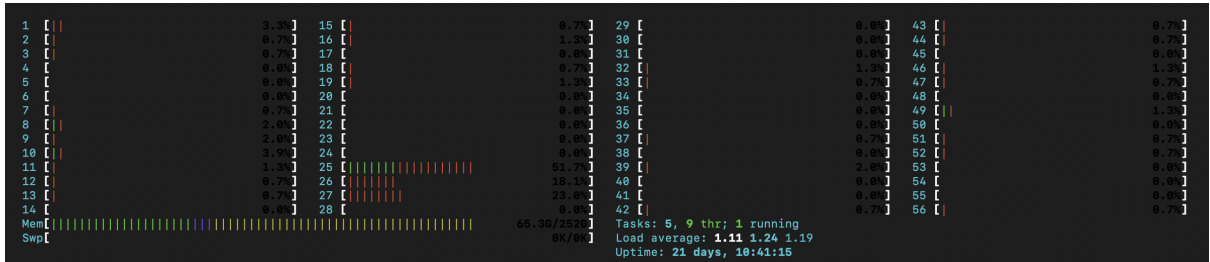


Figure 3.4: Visual representation of where work is being done on the Supercloud nodes. Notice all the computation work is clustered around nodes 25, 26, and 27, which are physically near each other on the processing chip.

Using eight processes across four nodes results in the best performance, closely followed by eight processes across a single node. Each worker in these two cases computes one build of the model, and optimizes it for two different parameter values. This illustrates that the time to communicate across nodes is expensive and we must have enough operations to amortize this cost to gain any improvement in speed. There is still a speedup by over a factor of 3 due to the parallelization, showing superior performance to a serial implementation. Note that for the full scale of the problem, the number of workers and nodes will need to be scaled according to the

# of Workers	Number of Nodes		
	1	2	4
1	149.84 s	-	-
2	97.35 s	96.35 s	-
4	64.75 s	59.16 s	69.53 s
8	49.75 s	50.95 s	46.49 s

Table 3.2: Execution time for eight MIP's given various numbers of works across various numbers of nodes.

problem size. These experiments show that parallelization is a good idea for performance and help us understand what to expect when we scale to larger problem sizes.

Chapter 4

Conclusion

We described a discrete optimization algorithm for instances of the SMTI problem in the context of school choice. This resulted in significant improvements upon current state-of-the-art methods in terms of performance and tractability. We introduce a new optimization model for SMTI, the first of its kind to balance cost as an objective. Using sparse arrays, cutting planes, integer relaxations, Julia code optimizations, and trade-off assessments, we are able to solve large scale instances of these types of problems as demonstrated in the computational experiment with BPS. Using a detailed parallelization strategy, we show strong performance increase that benefits from multiple processors.

We are able to show significant cost improvements over Boston Public School’s current Gale-Shapely stability style assignment mechanism. This 8-17% improvement in cost does not sacrifice the preferences of students.

4.1 Future Research

Future work not considered in this paper is the concept of robustness. Boston Public Schools is a large district with many schools of varying strengths and weaknesses, so families may have a difficult time of correctly developing a full list of preferences. Families are given a large menu of school options and are asked to rank them accurately. Given the large number of covariates they must consider in making their decisions, Kahneman and Tversky [15] tell us that students may not be confident and correct in their rankings. Their experiments in prospect theory reveal that individuals do not do a satisfactory job communicating their true preferences, and oftentimes express conflicting preferences. There is also evidence that shows the presentation of the information on schools effects which schools families determine to be "quality".

One of the objectives of Robust Optimization (RO) aims to address problems under uncertainty, such as this instance. Instead of using probability theory to describe uncertainty, RO uses uncertainty sets to transform the underlying stochastic optimization problem to a deterministic one. The PCF² model improves when we robustify it against sources of potential uncertainty, so in the future we could address preference uncertainty based on behavioral economics literature.

To do this, we must first define an uncertainty set on the data over which the solution is protected. This means that the true value of the family’s preference may fall within a range of the nominal value, where the range is determined by the architecture of the uncertainty set.

We want to make a trade-off between "full" robustness and the size of the uncertainty set: the choice of a large set results in a value never being too extreme for the optimization, but there is only a small chance that the parameter takes the worse case (i.e. preference #1 under no uncertainty is actually preference #20). A good uncertainty set choice would be similar to an ellipsoidal uncertainty set seen below that is convex in V . Then, student rankings could be robustified through a shuffling mechanism $r_{ij} = \bar{r}_{ij} + v_{ij}$ where

$$V_\epsilon = \{v_{ij} : v_{ij} \in \{0, 1\}, \|\mathbf{v}\| \leq \sqrt{2\ln(\epsilon)n}\} \quad (4.1)$$

The mechanism works in such a way that if a student ranks a school first, that school is allowed to take on a ranking of 2 or 3, while a school ranked second can become a 1 or a 3. Here, ϵ can be thought of as a risk-level parameter that increases or decreases the amount of students that have their preferences shuffled. An $\epsilon = 0.1$ means that 10% of students are shuffled.

Also not considered in this paper is to include a concept of neighborhood cohesion within the model framework. In the model presented, students living in the same neighborhood could very well be assigned to different schools. Crime rates have been shown to decrease when there is a sense of cohesion in the neighborhood, so we may want to encourage neighbors to attend the same school. If this model is to be implemented, it will require serious change on behalf of BPS officials. As we saw with the Boston Public Schools School Bus Routing algorithms, implementation of such a big change can be challenging. To overcome this, in future models we can limit the number of families who would be assigned to a school other than their current assignment. In this way, change can be incrementally increased until the full benefits of our PCF² model can be implemented.

Appendix A

BPS Budget

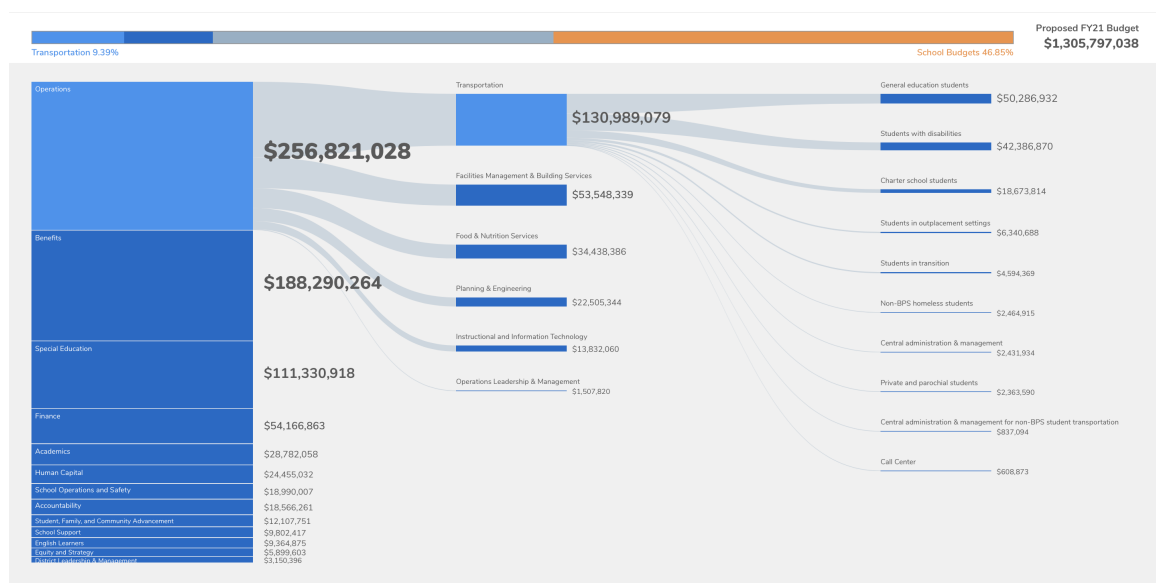


Figure A.1: Share of BPS budget going to transportation costs [2].

Appendix B

Additional Results

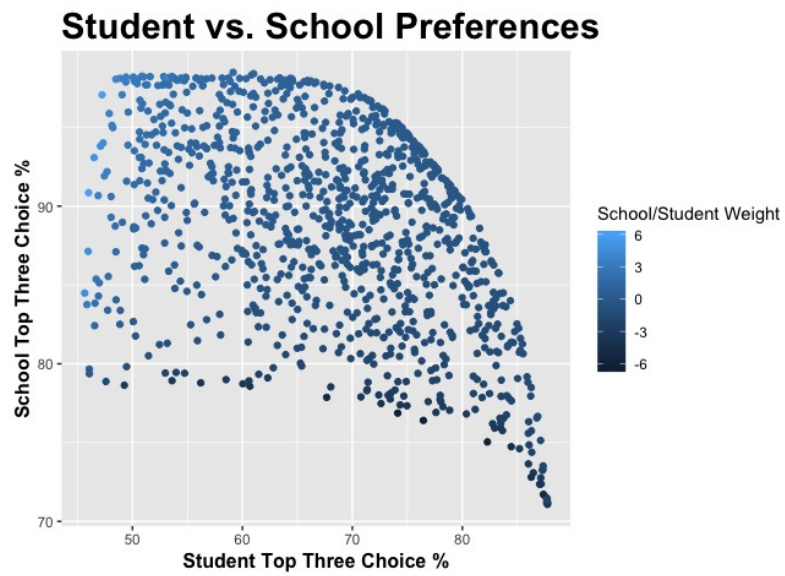


Figure B.1: Trade-off between school and family preferences.

Bibliography

- [1] Julia Documentation. <https://docs.julialang.org/en/v1/>.
- [2] Massachusetts school and district profiles. <http://profiles.doe.mass.edu/profiles/>.
- [3] Stable Matching: Theory, evidence, and practical design. <https://www.nobelprize.org/uploads/2018/06/popular-economicsciences2012.pdf>.
- [4] MIT Supercloud. <https://supercloud.mit/>, Feb 2020.
- [5] A. Abdulkadiroğlu, Y.-K. Che, and Y. Yasuda. Expanding "choice" in school choice. *American Economic Journal: Microeconomics*, 7(1):1–42, February 2015.
- [6] A. Abdulkadiroğlu and T. Sönmez. School choice: A mechanism design approach. *American Economic Review*, 93(3):729–747, June 2003.
- [7] P. C. Belford and H. D. Ratliff. A Network-Flow Model for Racially Balancing Schools. *Operations Research*, 20(3):619–628, June 1972.
- [8] D. Bergman, A. A. Cire, A. Sabharwal, H. Samulowitz, V. Saraswat, and W.-J. van Hoeve. Parallel combinatorial optimization with decision diagrams. In H. Simonis, editor, *Integration of AI and OR Techniques in Constraint Programming*, pages 351–367, Cham, 2014. Springer International Publishing.
- [9] D. Bertsimas, A. Delarue, and S. Martin. Optimizing schools' start time and bus routes. *Proceedings of the National Academy of Sciences*, 116(13):5943–5948, 2019.
- [10] A. L. Bodoh-Creed. Optimizing for distributional goals in school choice problems. *Management Science*, 0(0):null, 0.
- [11] M. Delorme, S. García, J. Gondzio, J. Kalcsics, D. Manlove, and W. Pettersson. Mathematical models for stable matching problems with ties and incomplete lists. *European Journal of Operational Research*, 277(2):426–441, 2019.
- [12] B. Dimitris and R. Weismantel. *Optimization Over Integers*. Dynamic Ideas, Belmont, MA, 2005.
- [13] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

-
- [14] K. Iwama, S. Miyazaki, Y. Morita, and D. Manlove. Stable marriage with incomplete lists and ties. In J. Wiedermann, P. van Emde Boas, and M. Nielsen, editors, *Automata, Languages and Programming*, pages 443–452, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.
 - [15] D. Kahneman and A. Tversky. Prospect theory: An analysis of decision under risk. *Econometrica*, 47(2):263–291, 1979.
 - [16] J. Kepner and H. Jananthan. *Mathematics of Big Data: Spreadsheets, Databases, Matrices, and Graphs*. The MIT Press, 2018.
 - [17] J. T. Linderoth and M. W. Savelsberg. *Topics in Parallel Integer Optimization*. PhD thesis, USA, 1998.
 - [18] D. Manlove, G. O’Malley, P. Prosser, and C. Unsworth. A constraint programming approach to the hospitals / residents problem. pages 155–170, 05 2007.
 - [19] D. Martinez, R. Bond, and M. Vai. *High performance embedded computing handbook: A systems perspective*. 01 2008.
 - [20] K. McKiernan. City council rips transportation costs in BPS budget, Nov 2018.
 - [21] D. Múnera, D. Diaz, S. Abreu, F. Rossi, V. Saraswat, and C. Philippe. A local search algorithm for smti and its extension to hrt problems. In *None*, 04 2015.
 - [22] A. Podhradsky. Aproximativní algoritmy pro problém stabilního párování [online]. Master’s thesis, Masaryk University, Faculty of Informatics, Brno, 2011 [cit. 2020-04-20].
 - [23] C. Rackauckus. Parallel computing and scientific machine learning course notes, 2019.
 - [24] T. Ralphs, Y. Shinano, T. Berthold, and T. Koch. *Parallel Solvers for Mixed Integer Linear Optimization*, pages 283–336. Springer International Publishing, Cham, 2018.
 - [25] A. E. Roth, U. G. Rothblum, and J. H. Vande Vate. Stable matchings, optimal assignments, and linear programming. *Mathematics of Operations Research*, 18(4):803–828, 1993.
 - [26] J.-C. Régin and A. Malapert. *Parallel Constraint Programming*, pages 337–379. Springer International Publishing, 2018.
 - [27] P. Shi. Guiding School-Choice Reform through Novel Applications of Operations Research. *Interfaces*, 45(2):117–132, April 2015.
 - [28] V. Strauss. Three big problems with school ‘choice’ that supporters don’t like to talk about, May 2017.
 - [29] C.-P. Teo and J. Sethuraman. The geometry of fractional stable matchings and its applications. *Mathematics of Operations Research*, 23(4):874–891, 1998.
 - [30] J. H. Vande Vate. Linear programming brings marital bliss. *Oper. Res. Lett.*, 8(3):147–153, June 1989.

- [31] B. Wilkinson and M. Allen. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. Pearson, New York, NY, 2005.