

Local Search Algorithm for University Class Schedule Optimization

The University of British Columbia
MATH 441
November 27, 2015

Knill, Stephanie
54882113

Valdivieso, Mauricio
25216145

Zhu, Ian
46888129

Contents

1	Background	1
1.1	University Class Schedule	1
1.2	The Local Search Algorithm	3
2	Methodology	4
2.1	Calculating Walking Distance	4
2.2	Our Local Search Algorithm	5
3	Results	9
4	Discussion	11
4.1	Efficiency of Local Search Algorithm	11
4.2	Sensitivity Analysis	12
5	Thoughts on the Project	15
	Bibliography	17

1 Background

While the University of British Columbia (UBC) is committed to “[providing] support and programming initiatives designed to remove barriers for students with disabilities and facilitate disability related accommodations for members of the UBC Vancouver community” [1], the onus is still upon the students to optimize their time at University, given their own unique and diverse set of constraints. In late August of the year 2015, a member of our group befell a most grievous injury that relegated her to the mercy of crutches. Weeks later when the subsequent winter term started, she was unable to attend her classes—those she did attend she was thoroughly late for¹ and was only pardoned from the walk-of-shame that befalls all latecomers due to her pitiable state—owing to the expansive, sprawling nature of the UBC Vancouver campus. Deeply concerned for our comrade’s well-being, we wanted to do something to alleviate her plight. What if this had been more than a temporary state? How would she be able to finish her degree whilst still maintaining a competitive enough grade point average (GPA) to fulfill her childhood dreams of mathematics graduate school? So we took her course schedule and minimized the walking distance between classes.

Or in Ian’s words, we optimized laziness.

1.1 University Class Schedule

As a second year undergraduate majoring in mathematics at UBC, there are a number of required (critical) courses for her to be eligible for promotion to third year (MATH 200, 215, 220, 221 and CPSC 210²), as well as 15 credits (5 courses) of electives. Additionally, many of these courses have pre-requisite and co-requisite requirements:

- MATH 215, 340: requires MATH 221
- MATH 300, 302, 317: requires MATH 200
- MATH 215: co-requires MATH 200
- MATH 300: co-requires MATH 317

Thus, MATH 200 and 221 must be taken in the first term while MATH 215 must be taken in the second term. If either MATH 300, 302, 317, 340 are taken as electives, they will be taken in term two. Using the data provided on the UBC Student Service Centre (SSC) for the 2015-2016 Winter Session, a table of the class locations, time, and term(s)

¹A conspiracy theory of her trying to trip a mathematics professor in an attempt to “crutch him up” was also in rampant circulation.

²MATH 210 or CPSC 210 may be taken to fulfill the mathematical computing requirement. We would like to take more computer science courses in upper years, however, so we will be taking CPSC 210 instead.

offered for all sections of the critical classes (Table 1) and desired electives (Table 2) was constructed.

Critical Classes

	Building	Room	Days	Time	Section	Term
MATH 200	LSK	201	MWF	9:00 - 10:00	101	1
	LSK	201	MWF	11:00 - 12:00	102	1
	ANNEX	1100	MWF	11:00 - 12:00	103	1
	BUCH A	104	MWF	13:00 - 14:00	104	1
	BUCH A	201	TTh	9:30 - 11:00	105	1
	BUCH A	104	TTh	15:30 - 17:00	107	1
MATH 221	LSK	201	MWF	10:00 - 11:00	102	1
	MATH	100	MWF	13:00 - 14:00	103	1
	LSK	201	MWF	13:00 - 14:00	104	1
MATH 220	ANNEX	1100	MWF	12:00 - 13:00	101	1
	LSK	200	MWF	10:00 - 11:00	102	1
	LSK	460	MWF	10:00 - 11:00	103	1
MATH 215	ANNEX	1100	MWF	10:00 - 11:00	201	2
	BUCH A	201	MWF	9:00 - 10:00	202	2
CPSC 210	WOOD	101	MWF	12:00 - 13:00	101	1
	HUGH	310	MWF	14:00 - 15:00	102	1

Table 1: Table of class locations, time, term(s) offered for all sections of critical classes.

Although on paper math majors experience a relatively large amount of freedom in choosing electives and math courses to fulfill graduation requirements, the competitiveness of graduate school and the prospective job market add further constraints. In an effort to maximize GPA, Calculus IV (MATH 317) will be taken in the second term in the hopes of still remembering materials from Calculus III (MATH 200) in first term. Calculating one's probability of getting into graduate school has also been deemed an asset—Introduction to Probability (MATH 302) is a much needed wake-up call. Likewise, it has been rumoured that the MATH 441 professor has nice math shirts; the pre-requisite Introduction to Linear Programming (MATH 340) must also be taken. Due to a childhood rumination with plant taxonomy, Weed Science (BIOL 317) is also essential.

Given these constraints for graduation requirements (required courses, credit requirements, pre-requisites & co-requisites), personal preferences, and different sections offered, our goal is to write a program that yields a class schedule that minimizes walking distance.

Elective Classes

	Building	Room	Days	Time	Section	Term
MATH 300	ANNEX	1100	TTh	14:00 - 15:30	201	2
	ANNEX	1100	MWF	13:00 - 14:00	202	2
MATH 302	LSK	201	MWF	11:00 - 12:00	201	2
MATH 317	LSK	460	MWF	14:00 - 15:00	201	2
	BUCH A	202	MWF	12:00 - 13:00	202	2
MATH 340	MATH	104	TTh	9:30 - 11:00	201	2
	BUCH A	103	MWF	12:00 - 13:00	202	2
BIOL 317	MCML	160	MWF	9:00 - 10:00	101	1

Table 2: Table of class locations, time, term(s) offered for all sections of elective classes.

1.2 The Local Search Algorithm

Rather than utilizing an exhaustive search method, the local search algorithm was favoured due to its relative computational efficiency. The local search algorithm attempts to find the optimal solution to a given optimization problem through many small, local changes. Initially given a feasible solution, each successive iteration makes a local or relatively simple change to the current solution, thereby creating another feasible solution. This cycle continues until either a time bound (i.e. number of iterations) has been met or when a solution is considered optimal (i.e. the best obtainable solution in the neighbourhood cannot be improved).

In the procedure known as “hill-climbing”, the local search algorithm picks the best neighbouring solution or any solution that is an improvement of the current solution. Although this method guarantees that each new solution is at least as good as the previous solution, one may arrive at a local optimal solution—not the global optimal solution. Our algorithm addresses this problem. Specifically, our code “randomly” produces a nearby feasible schedule and sets this as the current schedule. Thus, we are not bound to any local path and the entire search space may be explored with enough iterations.

2 Methodology

2.1 Calculating Walking Distance

In order to calculate the distance between any two locations on campus, a cartesian coordinate system in \mathbb{R}^2 was mapped onto the UBC Vancouver campus (Figure 1). Utilizing the Global Positioning System (GPS) provided by Google Maps, each square has side length of 50 metres.

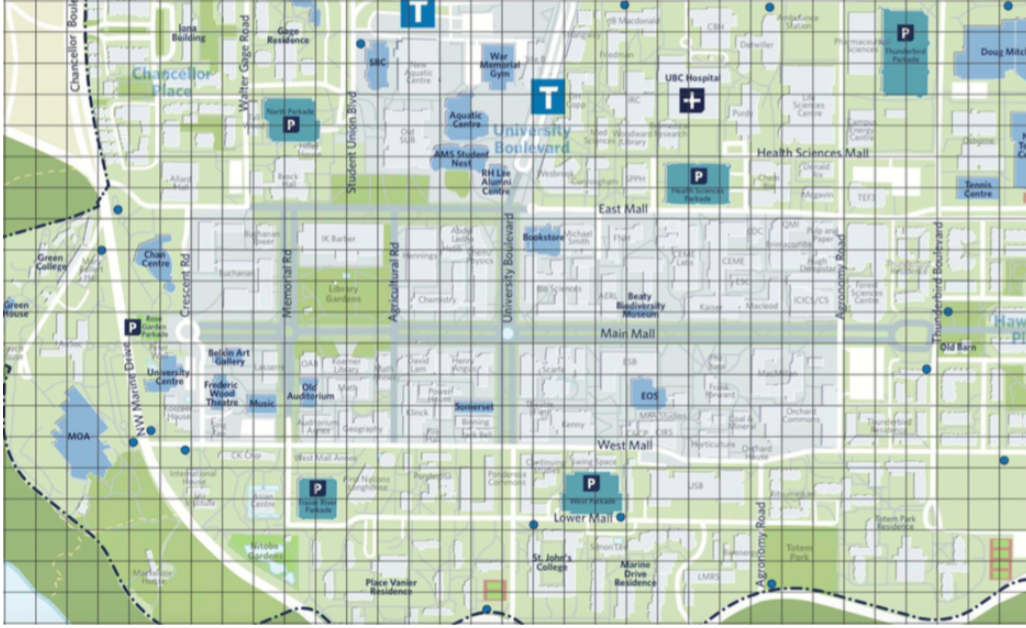


Figure 1: Cartesian coordinate map of the University of Building Construction (UBC), with origin set as bottom left coordinate. Each unit increment is 50m.

Using this we were able to generate a unique (x, y) coordinate for each classroom building and the bus loop (Table 3). Due to the excessive presence of blue fences on campus, the L^1 -norm was favoured over the Euclidean-norm in calculating the distance between classes. On a more serious note, crutches' lack of mobility (inability to cut corners, traverse grass), the grid-like nature of UBC pathways, and the modern day infeasibility of walking diagonally through building walls were also weighty factors in this decision.

Building	x	y	Building	x	y
ANNEX	12	8	IKB	11	12
BUCH A	8	11	LSK	13	12
BIOL	17	11	MATH	11	8
CHEM	14	10	MCML	20	12
HUGH	26	12	WOOD	20	16
ICICS	26	10	Bus Loop	13	19

Table 3: (x, y) coordinate for buildings on UBC Vancouver campus.

2.2 Our Local Search Algorithm

Our local search works by taking an initial feasible schedule (consisting of classes C_1, C_2, \dots, C_{10}) and performing the following procedure for a specified number of iterations. A single iteration can be broken down into three steps (Figure 2):

1. A class C_i is randomly selected and removed from the feasible schedule
2. Amongst a subset of the bank (consisting of all periods for C_i), a period of C_i is randomly chosen to be placed back into the schedule. The period chosen may or may not be the same period of C_i that was removed from the initial feasible schedule.
3. Insert class C_i back into the schedule

Case 1: *the resulting schedule is feasible.* Since the schedule is feasible, the iteration is complete. The next iteration will use this resulting feasible schedule as its initial feasible schedule.

Case 2: *the resulting schedule is infeasible (the incoming C_i is in a time slot of an already existing period in the schedule)* The incoming period C_i replaces the conflicting class in the existing schedule. Since the schedule is still infeasible—we are missing one class—we return to step two. Now the subset of the bank refers to the class that was just replaced. This process will continue until a feasible schedule is reached (case 1), thus terminating the iteration³.

Under the assumption that the student begins and ends his or her day at the Bus Loop, the objective function value at each iteration is then computed using the L^1 -norm. By computing the walking distance after each iteration, we can determine if the new solution

³It is worth remarking that since we begin with a feasible schedule, every iteration will result in a feasible schedule—either the original feasible schedule or a new feasible schedule. In the unlikely case where there is only one feasible schedule, the iteration will always complete because it can return to its initial state; this still counts as a full iteration.

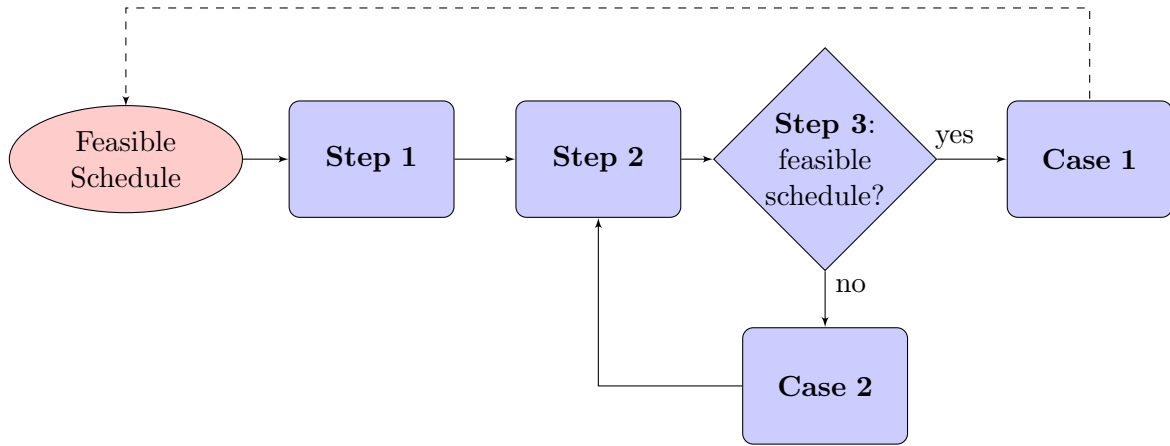


Figure 2: Flow chart illustrating algorithm of our local search function. Dotted arrow (not step in iteration) indicates that each iteration terminates with a feasible schedule.

represents an improvement over the previous best-found solution (the initial feasible schedule being the initial value to be improved upon). Since improvements are only stored, it becomes apparent that the last solution added to this list of solutions is the optimum solution.

Our (Java) code is divided amongst four main categories, known as packages (see Appendix A for full code):

1. **Model:** contains all the code necessary to solve the problem. This package includes our representation of Buildings, Periods, Schedules, the Solver, and the final output. This package will be analyzed in more detail after we discuss the remaining packages.
2. **Parsers:** contains the code critical to read the raw data used by the “Model” package. Class schedule data for the 2015-2016 winter session from the UBC SSC was transcribed into an Excel Spreadsheet; although this format makes inputting easy and intuitive for most users running this local search function, Java is not such person. Using an online data convertor [2], the raw data (buildings, periods, and critical class data) was converted into JSON format, which Java can easily interact with through our coded parsers. Due to the inefficiency of writing data directly into JSON format, this conversion method was favoured.
3. **Tests:** checks whether our code is behaving accordingly in each step, as well as the larger functions. This ensures every step is working as intended—we’re not just getting a lucky answer that looks correct.
4. **Tfi:** contains a bridge between the parsers and the raw data. The code in this package feeds the JSON data into the parsers to enable us to utilize our functions. All code

from this package was taken from the CPSC 210 2015 Project Mind the Gap Phase 1. Although this package is essential to our application, it is in no particular way related to our algorithm for local search.

Returning to the aforementioned “Model” package, there are several aspects of our code worth discussing to elucidate the inner workings of our algorithm and the project as a whole. A **Period** is defined by a class name, an x and y coordinate, a specific time, and a room number. The set of all periods is henceforth referred to as the **bank**. A **feasible schedule** is a subset of the bank that has every critical class exactly once and has no time conflicts (i.e. no two classes are taken at the same time). **Time Slots** are defined as a positive integer corresponding to a particular time period when a class at UBC takes place (Table 4). Our **Schedule Problem** is thus made up of the bank, a feasible schedule and a list of the names of critical subjects.

	Time Period	Time Slot (T1)	Time Slot (T2)
MWF	8:00-9:00	1	17
	9:00-10:00	2	18
	10:00-11:00	3	19
	11:00-12:00	4	20
	12:00-13:00	5	21
	13:00-14:00	6	22
	14:00-15:00	7	23
	15:00-16:00	8	24
	16:00-17:00	9	25
	17:00-18:00	10	26
TTh	8:00-9:30	11	27
	9:30-11:00	12	28
	11:00-12:30	13	29
	12:30-14:00	14	30
	14:00-15:30	15	31
	15:30-17:00	16	32

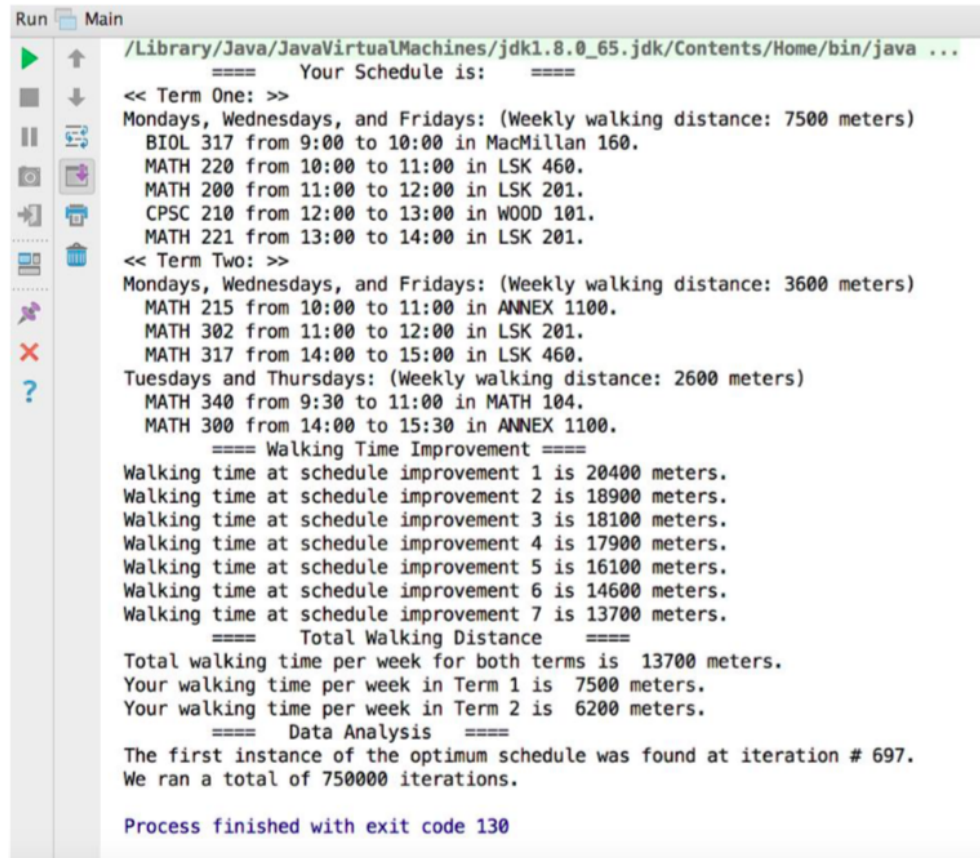
Table 4: Time Slots and their corresponding time periods for classes at UBC Vancouver in Term 1 (T1) and Term 2 (T2).

After completing the designated number of iterations, the optimal solution is outputted to the user. This solution consists of Periods, whose data type is composed of an x and y coordinate, a room number, and a time slot. With this information, the building where the class takes place is tracked down through the Parsers. The output to the user only conveys the essential information, namely the walking distances of the list of improved solutions and the optimal class schedule. Deeply concerned with the user’s potential mental breakdown

when presented with ominous lines of code, we further simplified our output by omitting any terms or days that did not have classes scheduled. If you are not taking classes on Tuesdays and Thursdays for example, these days would be omitted in the final output. Specification of the number of iterations and the raw data of the desired classes are all that is required for our program to output an optimal class schedule. Armed with this sense of bliss and utmost serenity, the user may now press the mythically proportional run button.

3 Results

Running our local search function in Java (Figure 3)



```
Run Main
/Library/Java/JavaVirtualMachines/jdk1.8.0_65.jdk/Contents/Home/bin/java ...
==== Your Schedule is: ====
<< Term One: >>
Mondays, Wednesdays, and Fridays: (Weekly walking distance: 7500 meters)
BIOL 317 from 9:00 to 10:00 in MacMillan 160.
MATH 220 from 10:00 to 11:00 in LSK 460.
MATH 200 from 11:00 to 12:00 in LSK 201.
CPSC 210 from 12:00 to 13:00 in WOOD 101.
MATH 221 from 13:00 to 14:00 in LSK 201.
<< Term Two: >>
Mondays, Wednesdays, and Fridays: (Weekly walking distance: 3600 meters)
MATH 215 from 10:00 to 11:00 in ANNEX 1100.
MATH 302 from 11:00 to 12:00 in LSK 201.
MATH 317 from 14:00 to 15:00 in LSK 460.
Tuesdays and Thursdays: (Weekly walking distance: 2600 meters)
MATH 340 from 9:30 to 11:00 in MATH 104.
MATH 300 from 14:00 to 15:30 in ANNEX 1100.
==== Walking Time Improvement ====
Walking time at schedule improvement 1 is 20400 meters.
Walking time at schedule improvement 2 is 18900 meters.
Walking time at schedule improvement 3 is 18100 meters.
Walking time at schedule improvement 4 is 17900 meters.
Walking time at schedule improvement 5 is 16100 meters.
Walking time at schedule improvement 6 is 14600 meters.
Walking time at schedule improvement 7 is 13700 meters.
==== Total Walking Distance ====
Total walking time per week for both terms is 13700 meters.
Your walking time per week in Term 1 is 7500 meters.
Your walking time per week in Term 2 is 6200 meters.
==== Data Analysis ====
The first instance of the optimum schedule was found at iteration # 697.
We ran a total of 750000 iterations.

Process finished with exit code 130
```

Figure 3: Java output for optimal class schedule. Objective function is 13,700 meters.

yields an optimal solution of 13,700 meters, which was found on iteration 697 of 750,000⁴. No classes in term 1 were scheduled on Tuesdays and Thursdays. To put this in more colourful terms, our beloved crutched team member excitedly created a worklist of her new and improved class schedule using the UBC SSC course scheduling (Figure 4).

⁴Each time the algorithm is run, the first instance of the optimal solution may be found on a different iteration.

Term 1 - 2015W (worklist "MATH441_Project")							
Legend:							
Registered Courses	Worklist Courses	Course Conflict	* days alternate weeks starting with Start Date	^ days alternate weeks starting with week following Start Date			
Sun	Mon	Tue	Wed	Thu	Fri	Sat	
09:00	BIOL 317 101 MCML - 160 (Sep 8-Dec 4)		BIOL 317 101 MCML - 160 (Sep 8-Dec 4)		BIOL 317 101 MCML - 160 (Sep 8-Dec 4)		
10:00	MATH 220 103 LSK - 460 (Sep 8-Dec 4)		MATH 220 103 LSK - 460 (Sep 8-Dec 4)		MATH 220 103 LSK - 460 (Sep 8-Dec 4)		
11:00	MATH 200 102 LSK - 201 (Sep 8-Dec 4)		MATH 200 102 LSK - 201 (Sep 8-Dec 4)		MATH 200 102 LSK - 201 (Sep 8-Dec 4)		
12:00	CPSC 210 101 WOOD - 6 (Sep 8-Dec 4)		CPSC 210 101 WOOD - 6 (Sep 8-Dec 4)		CPSC 210 101 WOOD - 6 (Sep 8-Dec 4)		
13:00	MATH 221 104 LSK - 201 (Sep 8-Dec 4)		MATH 221 104 LSK - 201 (Sep 8-Dec 4)		MATH 221 104 LSK - 201 (Sep 8-Dec 4)		
14:00							

Term 2 - 2015W (worklist "MATH441_Project")							
Legend:							
Registered Courses	Worklist Courses	Course Conflict	* days alternate weeks starting with Start Date	^ days alternate weeks starting with week following Start Date			
Sun	Mon	Tue	Wed	Thu	Fri	Sat	
09:00							
10:00	MATH 215 201 MATX - 1100 (Jan 4-Apr 8)	MATH 340 201 MATH - 104 (Jan 4-Apr 8)	MATH 215 201 MATX - 1100 (Jan 4-Apr 8)	MATH 340 201 MATH - 104 (Jan 4-Apr 8)	MATH 215 201 MATX - 1100 (Jan 4-Apr 8)		
11:00	MATH 302 201 LSK - 201 (Jan 4-Apr 8)		MATH 302 201 LSK - 201 (Jan 4-Apr 8)		MATH 302 201 LSK - 201 (Jan 4-Apr 8)		
12:00							
13:00							
14:00	MATH 317 201 LSK - 460 (Jan 4-Apr 8)	MATH 300 201 MATX - 1100 (Jan 4-Apr 8)	MATH 317 201 LSK - 460 (Jan 4-Apr 8)	MATH 300 201 MATX - 1100 (Jan 4-Apr 8)	MATH 317 201 LSK - 460 (Jan 4-Apr 8)		
15:00							

Figure 4: Optimal class schedule for term 1 and 2.

4 Discussion

Looking at our raw data, there are 6 sections of MATH 200, 3 sections of MATH 221, 3 sections of MATH 220, 2 sections of CPSC 210, and 1 section of our most beloved Weed Science (BIOL 317) in Term 1. This corresponds to **108** possible permutations. In Term 2, we have 2 sections of MATH 215, 2 sections of MATH 300, 1 section of MATH 302, 2 sections of MATH 317, 2 sections of MATH 340. This corresponds to **16** possible permutations.

Since each permutation in Term 1 can correspond with any permutation from Term 2, we have a total of **1,728** permutations. However, many permutations will not be feasible schedules—a feasible schedule must consist of classes that are not defined on the same period. Thus, the set of feasible schedules is a much smaller subset of the list of total permutations. Running our random schedule generator (the one used to generate our initial feasible schedule) exhaustively yields a total of **640** different feasible schedules.

4.1 Efficiency of Local Search Algorithm

Computationally, our local search on average finds the optimal feasible schedule after 1,000 trials. Whereas some may decide at this point to go in guns-a-blazing, arguing passionately for a more efficient exhaustive search method, one might want to hear us out before embarking on this rampant escalation.

While the project was still in its cradle stages, we identified a problem for our precious newborn: if we were to program our algorithm based off a greedy search, there would be a high probability of becoming stuck in a local solution. We also acknowledged, however, that a greedy search is highly efficient at producing near optimum solutions. But we wanted the best for our child—only the global optimum would do.⁵ So we addressed this potentially devastating ailment by programming a random, rather than hill-climbing, local search algorithm. This allowed for bad moves in order to cover the entire search space, thus eventually generating the global solution.⁶ Unfortunately, the results of our well-intentioned guidance resulted in an algorithm that was reminiscent of a less efficient exhaustive search method. Although both methods cover the entire sample space, our algorithm differs by also returning to previous solutions due to its random nature.

⁵Looking back now, a decent near-optimal solution may have for all intents and purposes be optimal. A pity the 15-year race lecture came too late.

⁶In retrospect, it would have been interesting to write out a greedy approach and compare this with our current algorithm. However, experience has told us that not sleeping—while questionably feasible—is most definitely not optimal. Needless to say this proposed route of analysis was swiftly vetoed by the IT department.

4.2 Sensitivity Analysis

Since there are many ways to measure the sensitivity of our algorithm, we decided to examine the most practical case: the effect of blocking a certain period (i.e. you cannot schedule any classes during a certain time period) on the objective function. That is, we examined how the minimum travel distance is affected when certain time periods are booked for various activities, such as math TA duties, club meetings, lunch, sanity breaks, or an aversion to 8 am classes. If the removal of a time period results in an infeasible schedule, then it cannot be blocked off. For example, Weed Science only has 1 section (Time Slot 1), so this time period cannot be removed.

Assumptions

Although we are blocking off a period for a specific purpose, we do not take into account the walking distance to the activity. While this may be the case if we book off a morning⁷ period to sleep-in, blocking off a class period to walk to the cafeteria for food does not increase the objective function. Had the location of the activity been taken into account, the true optimal schedule would most likely be different from our algorithm's outputted schedule.

Although it would be relatively simple to block off two or more periods at a time (in terms of programming, the only alterations to the code are done by deleting the classes specified at a certain time period), we did not include this in our analysis as the possible permutations would get quite out of hand.

Results and Analysis

The results of this analysis is summarized in Table 5. The largest increase in the objective function is when Time Slot 4 (MWF 11:00-12:00) is blocked.

In general, the optimal travel distance is not extensively altered from single period blocks. This is due to second year math courses having many sections. Similarly, most math classes are offered in the region known as the Math Triangle (Math, Math Annex, LSK)—a region where a number of math professors and students have disappeared under mysterious circumstances. Thus, removing a single period generally has minimal effect on the overall walking distance, with the notable exception being Time Slot 4. Blocking off this time slot

⁷Or afternoon

	Optimal Value	% Difference	Affected Classes
Term 1			
MWF 11:00-12:00	16100	17.518	MATH 200 - 3 left (BUCH)
MWF 12:00-13:00	14300	4.38	MATH 220 - 2 left (LSK) CPSC 210 - 1 left (HUGH)
MWF 13:00-14:00	14300	4.38	MATH 200 - 5 left MATH 221 - 1 left (LSK)
MWF 14:00-15:00	13700	0	CPSC 210 - 1 left (WOOD)
Term 2			
MWF 9:00-10:00	13700	0	MATH 215 - 1 left (LSK)
MWF 10:00-11:00	14000	2.19	MATH 215 - 1 left (BUCH)
MWF 12:00-13:00	13700	0	MATH 317 - 1 left (MATH) MATH 340 - 1 left (MATH)
MWF 13:00-14:00	13700	0	MATH 300 - 1 left (ANNEX)
MWF 14:00-15:00	15500	13.139	MATH 317 - 1 left (BUCH)
TTh 9:30-11:00	13800	0.73	MATH 340 - 1 left (BUCH)
TTh 14:00-15:30	13800	0.73	MATH 300 - 1 left (ANNEX)

Table 5: Sensitivity Analysis when a certain time slot is blocked. Percentage Difference defined as $\frac{\text{New Value} - \text{Optimal}}{\text{Optimal}} \cdot 100\%$. Affected Classes indicates the classes where a section(s) is removed and the remaining number of sections with their respective locations.

eliminates two sections of MATH 200 in LSK; the only remaining sections are in Buchanan⁸, hence outside of the Math Triangle.

Even in Term 2 where there are very few sections per math course, almost all of the remaining non-blocked sections are still within the Triangle, hence the minimal sensitivity. Again the exceptions are when the remaining sections are outside of the Triangle: Time Slot 19 (MWF 10:00-11:00) and Time Slot 23 (MWF 14:00-15:00). Interesting enough was the curious case of Time Slot 28 (TTh 9:30-11:00) where the remaining section of MATH 340 in Buchanan had a minimal effect on the objective function. Upon closer examination of the output, the new optimal schedule only had classes on MWF. Despite being forced to emerge from the vortex that is the Math Triangle, this was offset by the elimination of walking to and from the bus loop on two additional days.

⁸There is a fourth remaining section of MATH 200 located in LSK. However, it is in a time conflict with the only section of Weed Science (BIOL 317); this MATH 200 section was discounted leaving the remaining sections in Buchanan.

Further Extension

This sensitivity can be extended to other class schedules. If our dear crutch friend were to be tempted by the low heritage score of the Sauder building, then chances are that any blocks in her schedule will result in the same objective value since all business courses are offered in one building. Had she been in a combined or a more interdisciplinary major with a wider spread of class locations, then her schedule is likely to be more sensitive.

5 Thoughts on the Project

During the course of the project, our group was confronted with a number of constraints. Being the right-sizers we are, we decided to approach this as an optimization problem. Each member entered this course with varying backgrounds and experience; in order to produce the optimum paper, we needed to divide the labour in a way that maximized each of our unique comparative advantages. Thus, a list of constraints and relative information was tabulated for each group member:

Mauricio

- Currently taking CPSC 210 (main programming language is Java)
- Second year student majoring in Mathematics and Computer Science
- No experience in L^AT_EX

Stephanie

- Currently taking CPSC 110 (main programming language is drRacket; not powerful enough for a local search function)
- Originally majored in Environmental Sciences before switching to Mathematics (second year)
 - Experience and enjoyment in writing science papers
- Ill for 2 weeks during coding phase of the project
- No experience in L^AT_EX

Ian

- Taken CPSC 110 (main programming language is drRacket) and MATH 210 (main programming language is MATLAB)
- Research Assistant for Operations Research in Sauder
- Fourth year mathematics student with project management experience
- Some experience in L^AT_EX

Given these constraints, our optimal solution was a three department division: Mauricio as the IT department, Stephanie as the secretary/marketing/public relations, and Ian as upper management. Since Mauricio's interests and major lies within the realm of Computer Science, the optimal feasible solution consisted of him being the sole member of the IT department and the programming language for the algorithm would coincide with the language of CPSC 210. Due to Stephanie's illness during the coding phase, it would be

inefficient for her to learn a new language (Java), decipher the current code and then contribute new code. Rather, it seemed more sensible for her to learn L^AT_EX (a language she will eventually need to learn) and treat the Java code as a black box. Although Ian had no knowledge of Java, he does have experience in other programming languages and Sauder matters. This made him an ideal project manager—although he would not be able to write an entire local search algorithm in Java on his own, he could understand the code enough to manage the bigger picture, thus avoiding the programmer’s demise of tunnel vision. By having knowledge in both departments, this helped us address the drawback of specialization of labour: the divide between the user and the developers (i.e. the user tells the developer they want something but it’s not really possible). Due to Ian’s comprehension of the algorithm and the overbearing burden of writing the entirety of the code on Mauricio, the optimal division was where Ian analyzed the fruits of our search function.

Looking at this division of labour post hoc, it did prove to be optimal. In terms of time commitment, the division ended up being relatively uniform, with each member contributing in a field that their interest and speciality lay. Although our local search algorithm was less than ideal, we learned a lot about group work, which is an anomaly itself within an undergraduate mathematics degree. But most importantly the three of us have become akin to family. In an isolating degree like mathematics, having friends you can always count on is something words can’t even begin to describe.

After taking this course together, we have decided to take MATH 442 (Optimization in Graphs and Networks) together next term. One can only hope, however, that Professor Van Willigenburg has as good math shirts.

tl;dr You better not throw out our paper.

Bibliography

- [1] The University of British Columbia. (2015). Access and Diversity. *Student Services*. Vancouver, Canada. <<http://students.ubc.ca/about/access>> Last accessed on 23 November 2015.
- [2] Carter, S. (2015). Mr. Data Converter. *GitHub*. San Francisco, The United States of America. <<https://goo.gl/f8tO14>> Last accessed on 19 November 2015.