The Runtime Characteristics of Great Explorations' Prototype SAT-based Matching Algorithm

Kirk L. Lange

May 13, 2019

Abstract

Registrants of the Great Explorations event rank their top six preferred workshops, three of which they get to attend during the day's three workshop time-slots. My goal in this project was to write an algorithm that would assign each participant to as many of their most preferred workshops as possible. To this end, I choose to model this as a Boolean satisfiability (SAT) problem. Unfortunately, the exponential growth of the runtime makes realistically sized instances infeasible to solve.

In this report, I outline how and why I modeled this problem in terms of SAT, benchmark the SAT solver's speed at solving relatively small problem instances, then extrapolate the data to estimate how long it would take to solve a larger, real-world instance. It is my hope that this work will help inform future development of different kinds of matching algorithms for Great Explorations and other similar organizations.
`https://git.io/fjWl9`

1 Introduction

Great Explorations is a one-day science, technology, engineering, and mathematics (STEM) camp for 5th through 8th grade girls that takes place at Whitman College once every two years. Girls who register for this event get to rank their top six preferred workshops from a list of about thirty, three of which they get to participate in during the day's three workshop time-slots. Great Explorations is organized by a group of local volunteers who traditionally have had to do the advertising, participant registration, and workshop assignment all by hand through mail and paper. My senior capstone project team digitized and automated many of these processes by creating a registration website and an algorithm that assigns all the girls to as many of their preferred workshops as it can.

One path my capstone team did not get to explore was comparing the runtimes and outputs different kinds of algorithms would produce in solving the girls-to-workshops matching problem. At first glance it might seem like this problem most closely resembles bipartite matching. Yes, the girls and workshops can be organized into two independent sets of vertices G and W, respectively. Yes, a girl's preferences can be represented by connecting her vertex in G with edges directed to each appropriate vertex in W. Figure 1 even illustrates a small example like this when describing the model construction algorithm. Where this bipartite representation falls apart, however, is in how to expand the graph to account for the day's three time-slots. Moreover, there also needs to be a way for the event organizers to set limits on how many girls can get assigned to each workshop in order to avoid overcrowding.

The algorithm my capstone team came up with was iterative and greedy. While it did fulfill our client's requirements, it has a few shortcomings.

- Sometimes about 1% of the girls (out of 428 in this year's event) would get assigned to none of the workshops on their preference list.
- The girls that are randomly chosen to be the first in the iteration process are the most likely to get their top preferences.
- In order to ensure similar attendance levels between popular and unpopular workshops, we set *all* workshop capacities equal to their original capacity times the ratio between total girls and total available seats. Although equitable, this prevented many girls from getting assigned to their more preferred workshops even though there technically were spare seats.

It was my hope at the beginning of this project extension that a satisfiability (SAT)-based algorithm would provide the most optimal solution and solve the first two above issues of the greedy algorithm because of its more thorough search of the problem space. I also chose SAT because Boolean logic seemed like a fitting way to model this problem, as will be demonstrated in section 3. Originally I planned on directly comparing the workshop assignment outputs of the two algorithms, but unfortunately it proved infeasible to run the SAT solver on realistically sized problem instances. Instead, the focus of this technical report is to recount my process of modeling this problem in conjunctive normal form (CNF) and to examine the runtime characteristics of inputting increasingly large problem instances into the SAT solver.

2 Tools and Approach

For this project I chose to use Python and PycoSAT [3], a set of Python bindings for the pure C SAT solver PicoSAT. This allowed me to take advantage the ease of prototyping in Python while also reaping the efficiency benefits of using a library written in a lower-level language.

PycoSAT has two different SAT solver functions, one that returns the first solution it finds, and one that returns a list of all possible solutions. The former proved to be several orders of magnitude faster even on relatively small inputs and was therefore used instead of the latter throughout this project.

One convenience of PycoSAT that saved me a lot of hassle was its ability to solve general SAT instances, not just 3-SAT. This was tremendously helpful in developing the model construction algorithm because I could size and arrange the clauses in whatever way was easiest to interpret. Although the conversion from SAT to 3-SAT is fairly straightforward [2], not having to do so probably saved several hours of development time.

3 Model Construction Algorithm

The bulk of my work was in developing the model construction algorithm, that is, writing a function that would take as input the girls' preferences, workshop capacities, and number of time-slots, and output the Boolean CNF formula representing the problem. As a reminder, here is an example of what a Boolean formula in CNF looks like.

$$(x \vee y \vee z) \wedge (\bar{x} \vee \bar{y}) \wedge (y \vee z) \wedge (x)$$

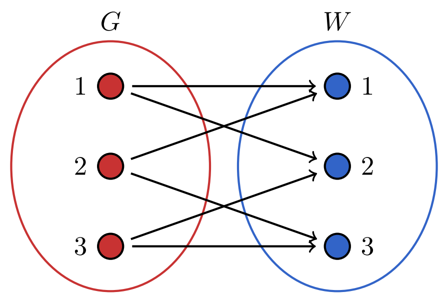
PycoSAT accepts for input this same CNF format, except represented as a list of lists of integers. Below is how the above formula would be represented in a Python program that uses PycoSAT.

$$[[1, 2, 3], [-1, -2], [2, 3], [1]]$$

So for the girls-to-workshop matching problem, how do we represent a problem instance in the above format? The first question I set out to answer was what would the Boolean variables themselves represent. After arriving at a sketch similar to Figure 1, I decided my solution would designate one variable for each possible edge that assigns a girl to a workshop.

Figure 1 illustrates some of this edge variable conversion process on a simple example model. This works exactly the same way as array index flattening. For this example, the formula for determining an edge's designated variable number is $v = g+(G(w-1))$, where v is the variable's number, g is the girl number, G is the total number of girls, and w is the workshop number. This can be expanded to account for time-slots as well, with $v = g+(G((w-1)+(W(s-1))))$, where W is the total number of workshops and s is the time-slot number. If the example in Figure 1 included two time-slots, we would additionally need the variables 10, 11, 13, 15, 17, and 18.

Continuing with the Figure 1 but with two time-slots example, all the missing variable numbers from 1 to the maximum possible variable number represent all the missing but otherwise possible edges that could go in the bipartite graph. These are the preferences that the girls did not list, for example, variable number 3 is not included because girl 1 did not list workshop 3 as one of her preferences. Because we do not want to assign a girl to a workshop she did not pick, these missing variables will appear in the CNF formula as negated variables, each within their own clause. Specifically, we must append $[[-3], [-5], [-7], [-12], [-14], [-16]]$ to the PycoSAT formula input.



Graph Edge Numbering

		Workshop		
		1	*2*	*3*
Girl	*1*	①	④	7
	2	②	5	⑧
	3	3	⑥	⑨

Vertices to Variables

Girl	*Preferences*	*Variables*
1	1, 2	1, 4
2	1, 3	2, 8
3	2, 3	6, 9

Figure 1: Example conversion from a bipartite graph to Boolean formula variables on a model with one time-slot, three workshops, three girls, and preference lists of size two.

To review, we now know that our formula will have some arrangement of clauses that include the variables 1, 2, 4, 6, 8, 9, 10, 11, 13, 15, 17, and 18, as well as these six exact clauses: $[[-3], [-5], [-7], [-12], [-14], [-16]]$. Before continuing, it is helpful to list all our problem's constrains:

1. Each girl should get exactly one workshop assignment per time-slot.
2. Each girl should not get assigned the same workshop twice.
3. No workshop should have more girls assigned to it than it has capacity.
4. No workshop should have fewer girls assigned to it than its capacity times the ratio between total girls and total available seats.
5. There must exist at least one solution.

We will ignore constraint 5 for now since it is not critical to the understanding of the core of the model construction algorithm.

These constraints are all similar in that they force certain groups of positive variables in our solution to be greater or lesser in number by a certain amount. With regards to constraint 1, only variable 1 or 4 can be positive in the final solution because girl 1 in time-slot 1 can only be assigned to either workshop 1 or 2. Similarly for constraint 2, only variable 1 or 10 can be positive because girl 1 cannot be assigned workshop 1 twice, once in time-slot 1 and then again in time-slot 2. For constraint 3, let us assume that each workshop is restricted to one girl per time-slot. In that case, out of variables 1, 2, and 3, only one can

Boolean CNF Formula $\geq$ and $<$ Patterns

Formula	*Required Positives*	*Subset Size*
$[[1, 2, 3]]$	≥ 1	3
$[[1, 2], [2, 3], [1, 3]]$	≥ 2	2
$[[1], [2], [3]]$	≥ 3	1
$[[-1, -2, -3]]$	< 3	3
$[[-1, -2], [-2, -3], [-1, -3]]$	< 2	2
$[[-1], [-2], [-3]]$	< 1	1

Figure 2: How the combinational arrangement of variables affects the required amount of positive variables in the solution in order to satisfy the formula.

be positive because only one girl can be assigned to workshop 1 in time-slot 1. Constraint 4 works much the same way, except it needs to calculate the total girls to total seats ratio beforehand. But the question still remains, how exactly does one exercise this level of precise control over the variables in a Boolean CNF formula?

Figure 2 demonstrates how the number of positive variables required to satisfy a formula is related to the variable combination subset (clause) size. For $\geq$ statements this is inversely proportional whereas for $<$ it is directly proportional. If I wanted to say, "Exactly two variables out of 1, 2, and 3 must be positive," I would need put the ≥ 2 and < 3 formulas together like so:

$$[[1, 2], [2, 3], [1, 3], [-1, -2, -3]]$$

Similarly, with this larger-scale example, "Exactly two variables out of 1, 2, 3, 4, and 5 must be positive," would translate to:

$$\begin{gathered}[[1, 2, 3, 4], [1, 2, 3, 5], [1, 2, 4, 5], [1, 3, 4, 5], [2, 3, 4, 5],\\ [-1, -2, -3], [-1, -2, -4], [-1, -2, -5], [-1, -3, -4], [-1, -3, -5],\\ [-1, -4, -5], [-2, -3, -4], [-2, -3, -5], [-2, -4, -5], [-3, -4, -5]]\end{gathered}$$

Now we all the pieces we need to run through the model construction algorithm and construct the final formula to be inputted into PycoSAT. Constraint 1 will contribute to the formula:

$$\begin{gathered}[[1, 4], [-1, -4], [10, 13], [-10, -13], [2, 8], [-2, -8],\\ [11, 17], [-11, -17], [6, 9], [-6, -9], [15, 18], [-15, -18]]\end{gathered}$$

Constraint 2:

$$[[-1, -10], [-4, -13], [-2, -11], [-8, -17], [-6, -15], [-9, -18]]$$

And constraint 3:

$$[-1, -2], [-4, -6], [-10, -11], [-13, -15], [-8, -9], [-17, -18]]$$

We must also not forget the six negated, single-literal clauses that disallow the missing edges of the graph from becoming part of our solution.

$$[[-3], [-5], [-7], [-12], [-14], [-16]]$$

Because the ratio between total girls and total available seats in this example is 1:1, constraint 4 appends this to the formula:

$$[[1, 2], [4, 6], [10, 11], [13, 15], [8, 9], [17, 18]]$$

Or in other words, at least one girl should be assigned to each workshop. In a situation where this ratio is 3:4 and the workshop capacities are 4, constraint 4 would add its model's ≥ 3 formula to the main formula.

Appending all five of these lists together gives us the formula PycoSAT needs to solve this problem. There are two solutions to this problem, but we are having PycoSAT only return the first one it comes across first for efficiency's sake. The two possible solutions are:

$$[1, -2, -3, -4, -5, 6, -7, 8, -9, -10, 11, -12, 13, -14, -15, -16, -17, 18]$$

and

$$[-1, 2, -3, 4, -5, -6, -7, -8, 9, 10, -11, -12, -13, -14, 15, -16, 17, -18]$$

It is easier to decipher the meaning of these solutions after translating the variable numbers back to their original girl number and workshop number pairs. Figure 3 does this conversion all the way back to the initial bipartite graph representation. PycoSAT's first listed possible solution has the left graph representing time-slot 1 and the right graph representing time-slot 2. Vice versa for the second listed solution.

And with that the walkthrough of the model construction algorithm is complete, except for constraint 5, "There must exist at least one solution." The only situation I discovered where a problem is not satisfiable is if any workshop is listed too few times throughout all the girls' preference lists. To overcome this, preference edges are edited as necessary before the model construction

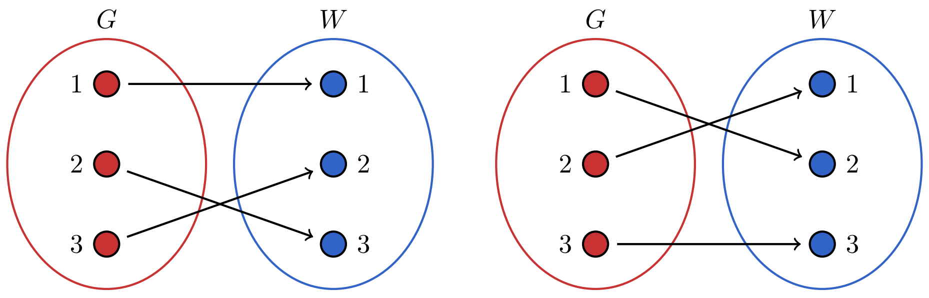


Figure 3: The two solutions to the example begun in Figure 1.

algorithm begins like so: for each under-preferred workshop, while it is under-preferred, visit the most popular workshop, then take a random girl's preference that points to that popular workshop and have it point to the current under-preferred workshop instead.

For more detail regarding any part of this algorithm, please refer to the source code in the GitHub repository linked here: `https://git.io/fjWl9`

4 Results

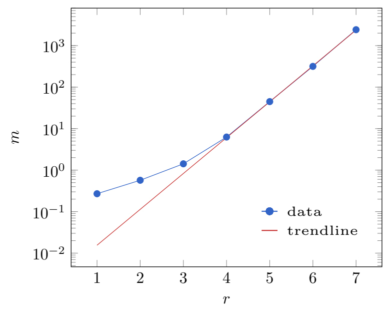
The purpose of this set of benchmarks is to help deduce the function $f(g,p,w) = t$, where g is the number of girls, p is the length of the girls' preference lists, w is the number of workshops, and t is the estimated amount of time it would take for PycoSAT to solve this large of a problem. To arrive at this function, I benchmarked the SAT solver on increasingly large model sizes, changing only one out of the three independent variables (g, p, and w) at a time. The number of time-slots s is not included as an independent variable and is assumed to be at a constant $s = 3$. This is because in real life the Great Explorations event is structured such that s cannot change. Throughout this benchmarking process, I measured the number of girls in terms of r, the ratio between the number of girls and the number of workshops. When it comes time to present the function f, r will be replaced with $\frac{g}{w}$.

Figure 4 shows how the runtime t increases as r and w increase. As is evident by the coefficients of determination R^2, the relation between w and t is positive linear, so $t = mw$. The slope m of this relation is marked in the

Runtime (ms)

$p = 3$	r						
w	*1*	*2*	*3*	*4*	*5*	*6*	*7*
6	1	2	5	33	222	1985	14332
7	1	2	6	33	254	2496	16055
8	1	3	7	39	304	2622	19122
9	1	3	9	44	368	3086	20957
10	2	4	10	50	411	3382	24195
11	2	4	11	58	448	3538	26876
12	2	5	13	62	491	4022	29222
13	3	6	14	71	526	4190	30524
14	3	6	16	86	592	4599	32747
15	3	7	18	82	612	4950	36381
m	0.27	0.57	1.42	6.27	44.80	316.42	2422.85
R^2	0.89	0.97	0.99	0.96	0.99	0.99	0.99

Figure 4: SAT solver mean runtimes, each out of ten samples, when varying r and w with constant $p = 3$.



(a) Logarithmic plot of runtime slope data when $p = 3$ with exponential trendline $m = (2.12 * 10^{-3})e^{1.99r}$.

$\mathbf{m = ke^{r(ap+b)}}$

p	$k * 10^3$	$ap + b$
3	2.12	1.99
4	3.68	2.39
5	5.13	2.63
6	2.30	3.06
slope	0.20	0.35
x-int	2.41	0.97
R^2	0.03	0.99

(b) The exponential growth of m in terms of r and p.

Figure 5: Determining the exponential growth of m by incrementing p.

second-to-last row of the table. However, m is not constant, and appears to have an exponential relation with r, i.e. $m \propto e^r$. Figure 5a plots this relation on a logarithmic graph along with a trendline as calculated by Google Sheets.

I repeated this process three more times for $p = [4, 5, 6]$ and recorded the trendline's coefficient and exponent in Figure 5b. This revealed that m is exponentially related to r as well as p, therefore $m \propto e^{rp}$. Using the data from Figure 5b we can further flesh out this relation into $m = (2.41 * 10^{-3})e^{r(0.35p+0.97)}$. Because k had such a low coefficient of determination, I am assuming it is constant and just using the x-intercept as k's constant value. The factor of the exponent that contains p, however, is clearly linear and expressed as such: $0.35p + 0.97$. Putting all these pieces together and returning to the question of what is $f(g, p, w)$, we arrive at:

$$t = (2.41 * 10^{-3})we^{\frac{g}{w}(0.35p+0.97)}$$

One issue I ran into was that for $p = [5, 6]$ and $r > 5$, the SAT solver refused to solve any problem instances. It did not output any errors or say that the problem was unsatisfiable, but instead it would just terminate for no apparent reason. Fortunately, I was still able to gather usable slope data for these values of p while $r \leq 5$ and include them in the Figure 5b table.

5 Discussion

Using the formula $f(g, p, w)$ we created in the previous section, we can determine the feasibility of running the SAT solver on realistic problems instances. We will define realistic as $g = [210, ..., 420]$, $p = 6$, and $w = 30$. (Remember, f has the built-in assumption that $s = 3$.) Figure 6 shows the estimates produced by the function.

Estimated Runtime

g	t (yrs)
210	0.00494
240	0.106
270	2.29
300	43.3
330	1062
360	22898
390	493273
420	10626039

Figure 6: Estimated SAT solver runtimes as predicted by $f(g, p, w)$. The runtime in days for the first two rows are 1.80 and 38.8.

Clearly it would be impractical to run the solver on the real-world data containing 428 girls as it would take over 10 million years on my desktop computer with an Intel i7-6700. However, problem sizes of half the real-world size and lower are feasible on a single machine. While not practical for the event organizers themselves, the SAT-based algorithm could potentially be compared to other algorithms on smaller problem instances. If for example a polynomial algorithm can create just as good of workshop assignments as the exponential, SAT-based algorithm on smaller problem instances, it might be the case that the polynomial algorithm would produce equally good assignments as the SAT solver's hypothetical output on real-world instances.

6 Future Work

One potential pitfall of this SAT-based algorithm in terms of the optimality of its output is in the series of steps it takes to satisfy constraint 5 as described in section 3, "There must exist at least one solution." The most optimal solution to this subsection of the problem would perform the fewest possible preference edits to the lowest ranking preferences when balancing out under-preferred workshops, and I am not convinced that my approach is the best.

Future work could also involve comparison against my capstone team's original iterative, greedy algorithm to sanity-check whether there even are measurable benefits in the workshop assignment output when using non-polynomial algorithms. I also never got to explore whether the Boolean CNF logic would be able to account for the ranked nature of the girls preference list. As of now, all the preferences are of equal weight. Whether it is during the workshop preference balancing phase or the choosing of which of the girl's preferences should be given to her, the inability to weigh a girl's top preference more heavily could be hurting the optimality of the SAT-based algorithm's output compared to the iterative, greedy algorithm which is able to factor in the list rankings. This, along with its infeasibility on realistic problem instances, is why I consider my SAT-based algorithm in its current state to be a prototype, rather than a fully-capable product, with much room for further improvement. Or, perhaps instead of trying to fix an inherently sluggish non-polynomial algorithm, efforts could be put into developing a new algorithm that combines the best of greedy, SAT, and maybe even some other completely different type of algorithm.

7 Acknowledgements

I would like to thank Dr. Janet Davis for her guidance throughout this project and for creating the LaTeX template of this technical report based on the CHI Proceedings Format [1].

References

[1] CHI. 2018. CHI proceedings format. `https://chi2018.acm.org/chi-proceedings-format/`. (2018).

[2] Sanjoy Dasgupta, Christos Papadimitriou, and Umesh Vazirani. 2008. Algorithms. In McGraw-Hill, New York, NY, USA. ISBN: 978-0-07-352340-8.

[3] Ilan Schnell. 2017. PycoSAT. `https://pypi.org/project/pycosat/`. (2017).