The LP relaxation of the maximum weight matching problem and its application on Covid-19 one-to-one help problem

Ran Liu rliu361@gatech.edu
903515184

1 Introduction

Graph matching is one of the most fundamental problems in computer science [4]. The maximum weight matching problem is a specific type of the graph matching problem. There are two types of the maximum weight matching problem: the exact maximum weight matching problem considers weighted graphs while the maximum cardinality matching problem considers unweighted graphs. Both of the maximum weight matching problems can be exactly solved in $\tilde{O}(m\sqrt{n})$ time [4]. The development of maximum weight matching algorithms has encouraged a lot of applications such as transportation problems and job assignment problems.

This project considers the weighted maximum weight matching problem with integer decision variables and its linear programming (LP) relaxation. Although integer programming (IP) would definitely give integer decisions, its computational cost increases significantly comparing with linear programming when the problem scale increases. There are some research works indicating that the linear programming relaxation of integer problems often provides integer optimal solutions [5, 3]. The use of linear programming relaxation on this problem could reduce the required computational cost and help people understand more about the effectiveness of the LP relaxation.

In this project report, I study the effectiveness of LP relaxation of the maximum weight matching problem by 1) characterising integer problems that have LP relaxations with integer solutions, 2) comparing the computational cost between integer problems and their LP relaxations, 3) performing extensive experiments to estimate the probability of LP relaxations giving integer decisions. After those basic experiments, I manually collected two Covid-19 infected patients dataset and applied the model on real-world data (introduced in 1.2), and challenged the model.

1.1 Problem setup

The notation and terminology for the maximum weight matching problem are introduced here to help understand this problem. This problem studies a weighted undirected graph $G = (V, E, w)$, where $2n = |V|$, $m = |E|$, and w, respectively, are the total number of vertices, the total number of edges, and the edge weights assignment where $w_{i,j}$ represents the (i, j) edge weight. In the maximum weight matching problem, the aim is to find a set of matching that maximizes the summed weights of the existing edges.

In this project, the specific problem studied only concerns when $2n = |V|$ is an even number. That is to say, the optimal integer solution would have all vertices considered. Also, the specific problem studied constrains that for all vertices, at most one edge in the matching may be incident on each vertex. Synthetic dataset with randomly generated weights ranged from 0 to 1 are used to theoretically test the model before applying models to real-world dataset.

1.2 Real-world application

The maximum weight matching problem can be applied to solve real-world assignment problems. In this project, I apply the model to solve the one-to-one help assignment problem during Covid-19. The motivation and the setup of this problem are introduced below.

The setup of this assignment problem is motivated by Chinese government's strategy to distribute resources during the pandemic of Covid-19. Being the first country that identified the virus, China applied a resources distribution policy called "one province helps one city" [1], which assigns 16 provinces to share resources including medical equipment and medical personnel to 16 cities in Wuhan, the epidemic's center, to combat the virus. The assignment matching aims to maximize the resources differences between the province that provides help and the city that accepts help to fully utilize resources. This becomes a bipartite maximum weight matching problem between 16 provinces and 16 cities. This policy is the motivation of this application [2].

I modified the problem to make it a maximum weight matching problem instead of a bipartite maximum weight matching problem. Assume that we want to distribute resources among 34 provinces of China, and thus we want to find a matching that pairs provinces to help each other with the objective to maximize the total differences between pairs. This becomes a maximum weight matching problem with $2n = 34$. Similarly, assume that we want to distribute resources among the 412 infected cities of China, this becomes a maximum weight matching problem with $2n = 412$. This is the setup of our problem that maximize the help between provinces and cities.

The total number of infected patients inside each area is used to quantify the resources deficiency of that area. That is to say, cities/provinces with large amount of infected patients are considered in need of more help, while cities/provinces with few infected patients are considered resourceful. I collected two dataset, the first dataset is a province-level dataset that contains numbers of infected patients in every provinces; the second dataset is a city-level dataset that contains numbers of infected patients in every cities. The differences between province-to-province/city-to-city pairs are calculated as model input. The model with real-world application is setup as above to compare model performance between real-world data and synthetic data.

The data collecting process and application results are concluded in Section 4.

2 Model statement

Definitions

Below definitions are universal among different models.

Parameter

– $2n$ is the total number of vertices, where n is an integer to ensure that the total number of vertices is an even number.

– T is the total trial size. Except for the last experiment in Section 5.2, which used a trial size of 10000, all other experiments used a trial size of 100.

Index sets

– $h = 1, 2, ..., 2n$ are indices for different vertices.

– $i = 1, 2, ..., 2n$ are indices for different vertices.

– $j = 1, 2, ..., 2n$ are indices for different vertices.

Model 1 – the basic model

This model is the basic formulation of the maximum weight matching problem.

Data

– $w_{i,j}$ forms the weight matrix W for weights of different edges of a complete graph on $2n$ vertices. Each $w_{i,j}$ is randomly generated from a uniform distribution $w_{i,j} \sim \mathcal{U}[0, 1]$. Since the graph is undirected, $w_{i,j} = 0$ for all $i \geq j$.

Variable

– $x_{i,j}$ is the decision variable that represents the matching. When the model is an integer model, $x_{i,j}$ is a binary indicator with two possible values $x_{i,j} = 0$ or $x_{i,j} = 1$, where $x_{i,j} = 1$ if edge (i, j) is in the matching. When the model is a LP relaxation version, $x_{i,j}$ is a float with lower bound 0 and upper bound 1. Since the graph is undirected, $x_{i,j} = 0$ for all $i \geq j$.

Objective

– maximize $\sum_{i<j} w_{i,j}x_{i,j}$. Maximize the total weights of the matching.

Constraints

– $\sum_{h=1}^{i-1} x_{h,i} + \sum_{j=i+1}^{2n} x_{i,j} \leq 1$ for every i. This means that at most one edge in the matching may be incident on vertex i for every i.

– $x_{i,j} \geq 0$ for every i and j.

– $x_{i,j} \leq 1$ for every i and j.

Model 2 – additional constraints

Model 2 is a variant of model 1 with additional constraints, which are the odd-set constraints for sets of cardinality 3. The additional constraints are written below and the rest of model 2 is the same as model 1.

Constraints

– $x_{i,j} + x_{j,h} + x_{i,h} \leq 1$ for all distinct $i < j < h$.

Model 3 – real-world dataset

Model 3 is the application of model 1 and model 2 on real-world dataset. The model remains the same but the definitions about data are changed below.

Data

– p_i is the total number of infected patients in area i, its unit: person. Note that since the unit of p_i is person, p_i is an integer according to its definition.

– $w_{i,j}$ is a matrix formed by p_i, where $w_{i,j} = |p_i - p_j|$ is the absolute difference between area i and area j.

Note: the selection of random seed

In all our experiments, we select random seeds in a fixed way to replicate the experiment results. For each trial of $k = 1, 2, ..., T$ times, I select the random seed as the sum of my student id $ID = 903515184$ and k. Thus, in the kth experiment, the random seed is $ID + k$. In other words, if the trial size is 100, the random seeds are from 903515185 to 903515285.

3 Model validation and experiments

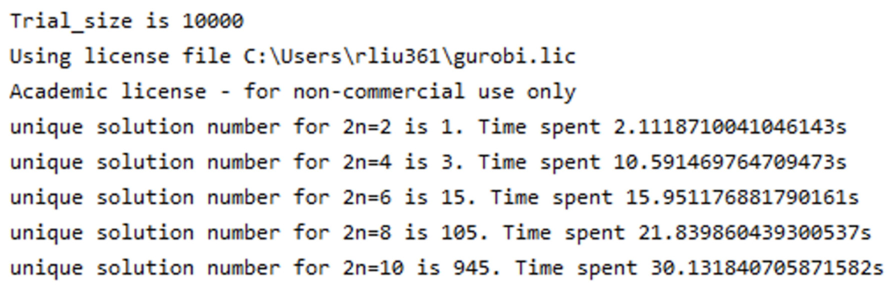
In this section, I first validate the integer version of model 1 to confirm that the model is correct. Some small-scale cases are performed and analyzed for both the integer version and LP relaxation of model 1 to validate the model. After that, large-scale cases are performed to examine the qualitative behaviors of model 1. After the validation of model 1, I added constraints that correspond to model 2 and checked their performance respectively.

3.1 Unique solution validation (model 1 IP)

The setup of the integer maximum weight matching problem determines that only limited amount of unique integer solutions exist. Obviously, through mathematical induction, we can prove that the amount of unique integer solutions $s_{2n} = s_{2n-2} \times (2n-1)$, where the initial point $s_2 = 1$, which corresponds to one possible assignment for a pair of vertices. Thus, it is easy to calculate that for $2n = 4, 6, 8, 10$, the total amount of possible unique solutions are listed as below:

$$s_4 = 1 \times 3 = 3, \quad s_6 = 3 \times 5 = 15, \quad s_8 = 15 \times 7 = 105, \quad s_{10} = 105 \times 9 = 945. \qquad (1)$$

Thus, I conducted a set of experiments with 10000 trials to validate this. The results are shown in Figure 1, where the total numbers of unique solutions for different $2n$ are correct.



```
Trial_size is 10000
Using license file C:\Users\rliu361\gurobi.lic
Academic license - for non-commercial use only
unique solution number for 2n=2 is 1. Time spent 2.1118710041046143s
unique solution number for 2n=4 is 3. Time spent 10.591469764709473s
unique solution number for 2n=6 is 15. Time spent 15.951176881790161s
unique solution number for 2n=8 is 105. Time spent 21.839860439300537s
unique solution number for 2n=10 is 945. Time spent 30.131840705871582s
```

Figure 1: Experiment log of counting unique solutions.

3.2 Small-scale cases validation (model 1 IP/LP)

To validate the model, relatively small vertices numbers ($2n$) are selected first. Vertices number $2n = 4$ and $2n = 6$ are chosen to run the experiments and the corresponding solutions are visualized. Cases when the LP relaxations give integer solutions and non-integer solutions and their corresponding IP solutions are listed together to gain an intuitive impression of the problem.

In the following, the upper triangulars without diagonal entries of the data matrix W, linear solution matrix X_l, and integer solution matrix X_i are visualized with different colors representing different entry values. The colorbar with range $[0, 1]$ is shown on the right and different entry values are printed inside the matrix block.

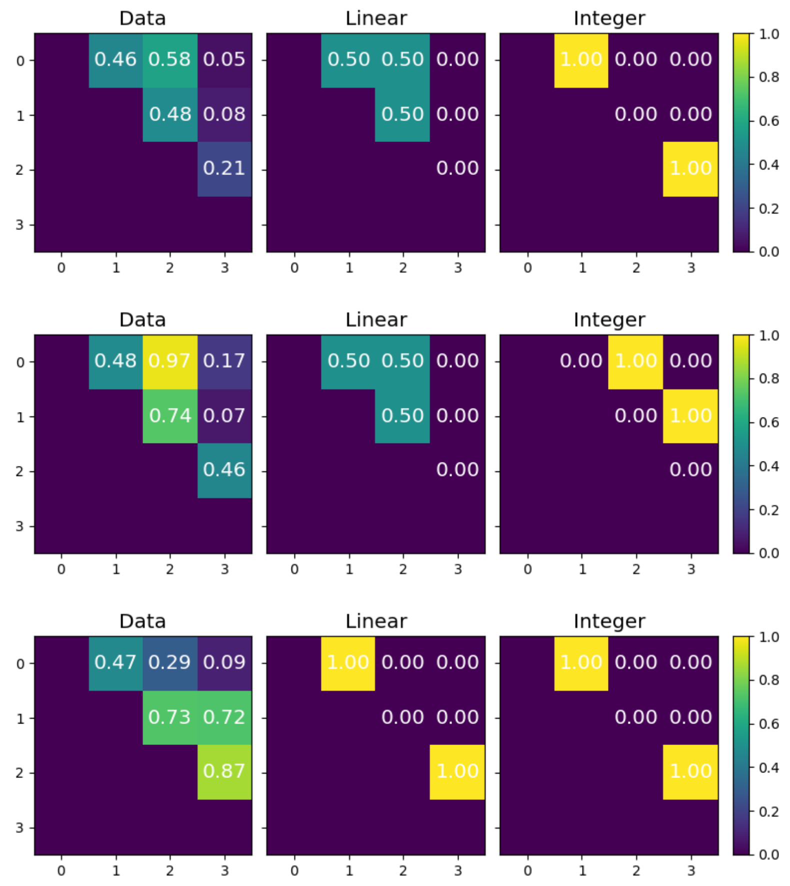


Figure 2: Three test experiments on $2n = 4$. The first matrix is the data matrix with the upper triangle filled with $w_{i,j}$ values. The second matrix is the linear relaxation of the question. The third matrix is the integer solution of the matrix. The first two examples give non-integer results while the third one gives the expected integer result.

2n = 4 Cases

Figure 2 shows the data matrices, the LP optimal solutions, and the integer optimal solutions of three different experiments. In the first two experiments, the LP relaxations do not give integer optimal solutions while in the third experiment, the LP relaxation gives integer optimal solution. This behaviour is analyzed in the following.

There are only 3 possible unique integer solutions for $2n = 4$ and the solution set can be written as $\mathcal{S} = \{\mathcal{S}_1 = (1\text{-}2, 3\text{-}4), \mathcal{S}_2 = (1\text{-}3, 2\text{-}4), \mathcal{S}_3 = (1\text{-}4, 2\text{-}3)\}$. In the first experiment, the optimal solution of the integer program is selected among the three possible integer solutions with optimal values $s_1 = 0.67, s_2 = 0.66, s_3 = 0.53$. Thus, the matching $\mathcal{S}_1 = (\text{1-2}, \text{3-4})$ is selected with the largest optimal value. In the LP relaxation of the first experiment, more combinations of edge selection become possible solutions, such as $\mathcal{S}_4 = (\text{1-2}, \text{1-3}, \text{2-3})$ with 0.5 as its edge weights or $\mathcal{S}_5 = (\text{1-2}, \text{2-3}, \text{3-4}, \text{1-4})$ with 0.5 as its edge weights. In this experiment, $\frac{1}{2}(w_{1,2} + w_{1,3} + w_{2,3}) = 0.76 \geq 0.67$, and thus $\mathcal{S}_4 = (\text{1-2}, \text{1-3}, \text{2-3})$ with 0.5 as its edge weights is selected as its LP optimal solution. Intuitively, it is obvious that the LP relaxation would always select 0.5 as its optimal solution in a set of cardinality 3 because of the following reason:

$$\text{compare } w_{i,j} \text{ with } w_{i,h} + w_{j,h} \begin{cases} \text{optimal solution is 0.5, if } w_{i,j} \leq w_{i,h} + w_{j,h} \\ \text{optimal solution is 1, if } w_{i,j} \geq w_{i,h} + w_{j,h} \end{cases} \tag{2}$$

This means that if the model decides to choose a set of cardinality 3, the optimal solution for this particular selection would either be 0.5 or would collapse to 1. The experiment results support this idea though I did not find a more precise and general mathematical proof for this specific problem.

Similarly, in the second experiment of $2n = 4$, the integer optimal solution is selected from $\mathcal{S} = \{\mathcal{S}_1 = (\text{1-2}, \text{3-4}), \mathcal{S}_2 = (\text{1-3}, \text{2-4}), \mathcal{S}_3 = (\text{1-4}, \text{2-3})\}$, which correspond to optimal values $s_1 = 0.94, s_2 = 1.04, s_3 = 0.91$, and thus the matching $\mathcal{S}_2 = (\text{1-3}, \text{2-4})$ is selected as the integer optimal solution. While the LP relaxation finds $\frac{1}{2}(w_{1,2} + w_{1,3} + w_{2,3}) = 1.095 \geq 1.04$ as its LP relaxation optimal solution.

The third experiment provides a different case when the LP relaxation gives integer optimal solutions. Through a manual check of all possible linear combinations with 0.5 as its weights (the reason is listed above), I found no solution better than the integer optimal solution. The above three experiments validate this model via examples.

2n = 6 Cases

Figure 3 shows two experiments when $2n = 6$. Similarly, the LP relaxation gives a better non-integer optimal solution via replacing the weights 1 with weights 0.5 while including more entries from the data matrix. A manual check of all possible combinations would prove the correctness of the model formulation.

3.3 Large-scale cases validation (model 1 IP/LP)

The small-scale cases presented above show that the model is acting normally through examples. In this subsection, we further validate the model by running some large-scale cases to examine model's qualitative behavior when $2n$ goes large. This validation is performed via the qualitative check of the computational cost of the model, and the probability of acquiring integer solutions through LP relaxations.

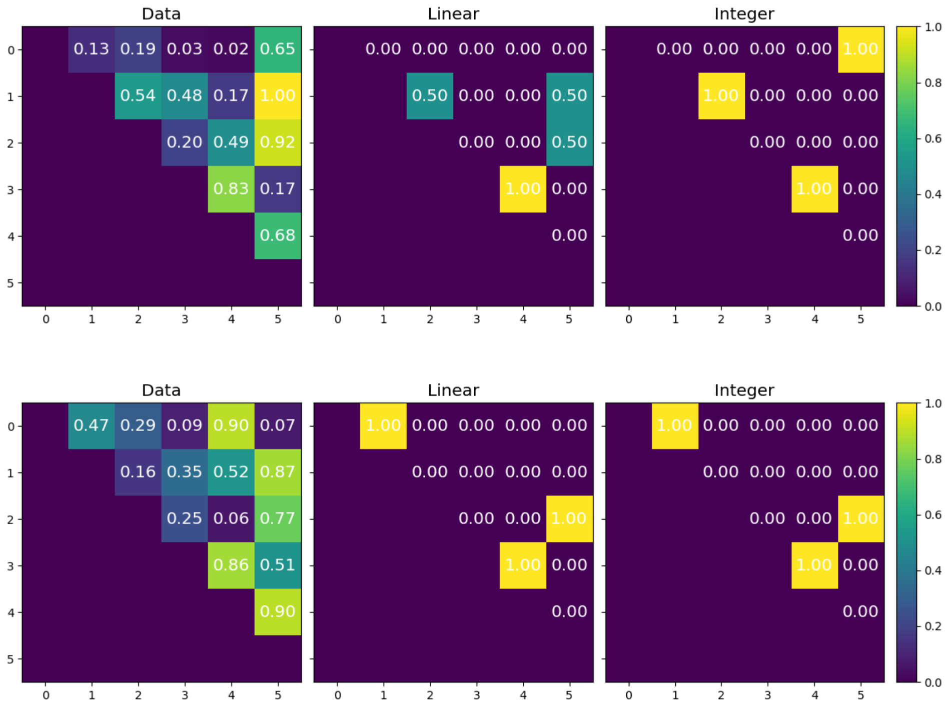


Figure 3: Two test experiments on $2n = 6$.

Computational cost analysis

The computational cost analysis is performed by: 1) comparing the computational cost between integer programming and its LP relaxation; 2) plotting the computational cost of conducting 100 trials on the LP relaxation of model 1. The experiment results are shown in Figure 4.

In the first figure, experiments on $2n = [100, 200, 400, 800, 1600, 3200]$ are performed and the logarithms of selected $2n$s are visualized on the x axis. The numbers on plot are the raw time differences between IP and LP. We can see that when the value of $2n$ is relatively small, the time difference is almost neglectable (only 5 seconds when $2n = 1600$, where the experiment time is around 150 seconds). However, when $2n$ goes big, the time difference increased significantly to 300 seconds with the experiment time around 2000 seconds. The LP relaxation could reduce the computational time by 15% when $2n$ is big enough. However, it is worth noting that the computational time varies a little with the selected random data matrix. Hence, this experiment can only show the possible promise of LP relaxation.

In the second figure, we show the computational cost of the LP version of model 1 on 100 trials when $2n$ goes large. I conducted experiments on $2n$ from 100 to 2000 with a step

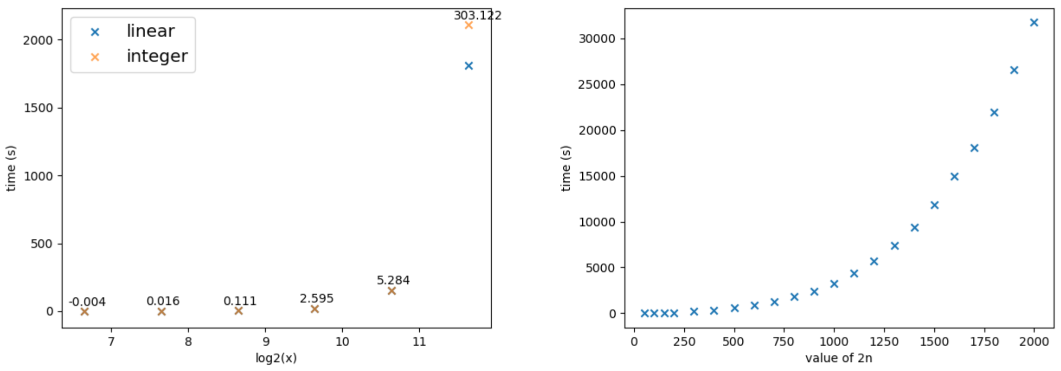


Figure 4: Computational cost analysis of model 1. The left-hand-side figure shows the computational cost difference between integer programming and its linear programming relaxation. The right-hand-side figure shows the computational cost of performing 100 trials on the linear relaxation of model 1.

Probability (Model 1 LP) (Selected data)					
2n	50	100	200	400	800
Probability	0.66	0.59	0.47	0.46	0.38

Table 1: Probability of having integer solutions in model 1 LP relaxation (selected numbers).

size of 100. We can see that the computational cost of LP relaxation explodes exponentially. The computational cost of conducing an experiment with large $2n$ is really expensive (30000 seconds is around 8 hours), that prevents me from performing larger experiments with more trials.

Probability of integer solutions

In Figure 5 and Table 1 we show the probability of having integer solutions in the LP relaxation of model 1, where Table 1 contains selected datapoints from Figure 5. We can see that the probability of having integer solutions drops almost linearly when the value of $2n$ increases, which is as expected. The probability is less than 50% when $2n \geq 200$. Note that we only conducted 100 trials to evaluate the probability to reduce computational time. This trend tells us that the LP relaxation of the basic setting (Model 1) of the maximum weight matching problem can still provide integer solutions effectively when the value of $2n$ goes large (a probability of 0.3 when $2n = 2000$). More constraints are expected to improve the model performance.

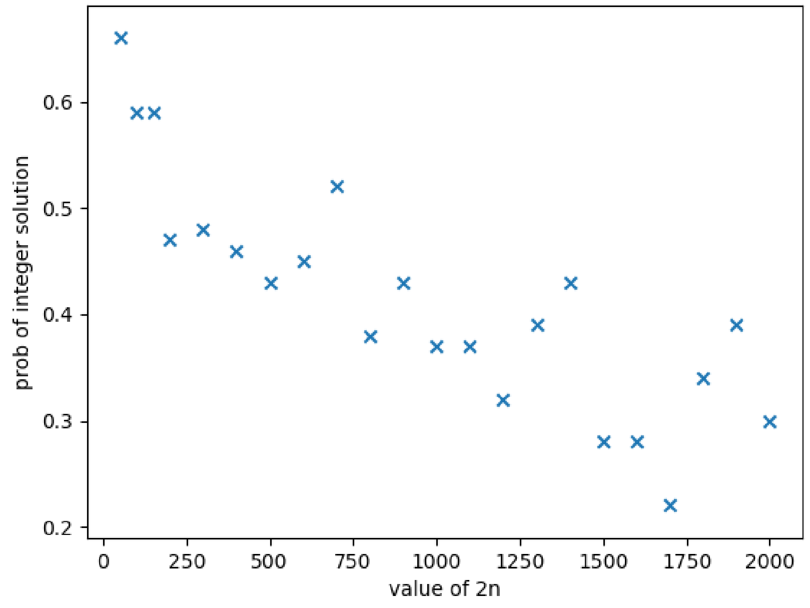


Figure 5: Probability of having integer solutions in model 1 LP relaxation.

3.4 Additional validation (model 2 LP)

After having the model validation and gaining the intuition about this problem in the previous parts, the additional constraints added in model 2 are examined. Intuitively, as observed in Figure 2(1)(2) and Figure 3(1), the non-integer solutions are generated by replacing two 1 datapoints with three 0.5 datapoints, and those situations can be removed by adding the additional constraint. Thus, we check if that is the case via the below analysis.

Bug fixing

I checked the non-integer solution examples and found that when having additional constraints, the model gives the optimal solution with less precision, where numbers such as 1.0000000000000002 and 0.9999999999999991 are concluded as non-integer solutions. This situation did not happen in model 1 before. I assume that this is a Gurobi precision problem, and this is out of the consideration of precision in our researched question. Thus, I lowered the precision of the code that determines whether if a solution is an integer solution to remove this small bug (so that $1.0000000000000002 = 1$ now). In Figure 7, datapoints collected before the bug is fixed and datapoints collected after the bug is fixed are displayed together via different colors. We can see that this bug is not affecting much.

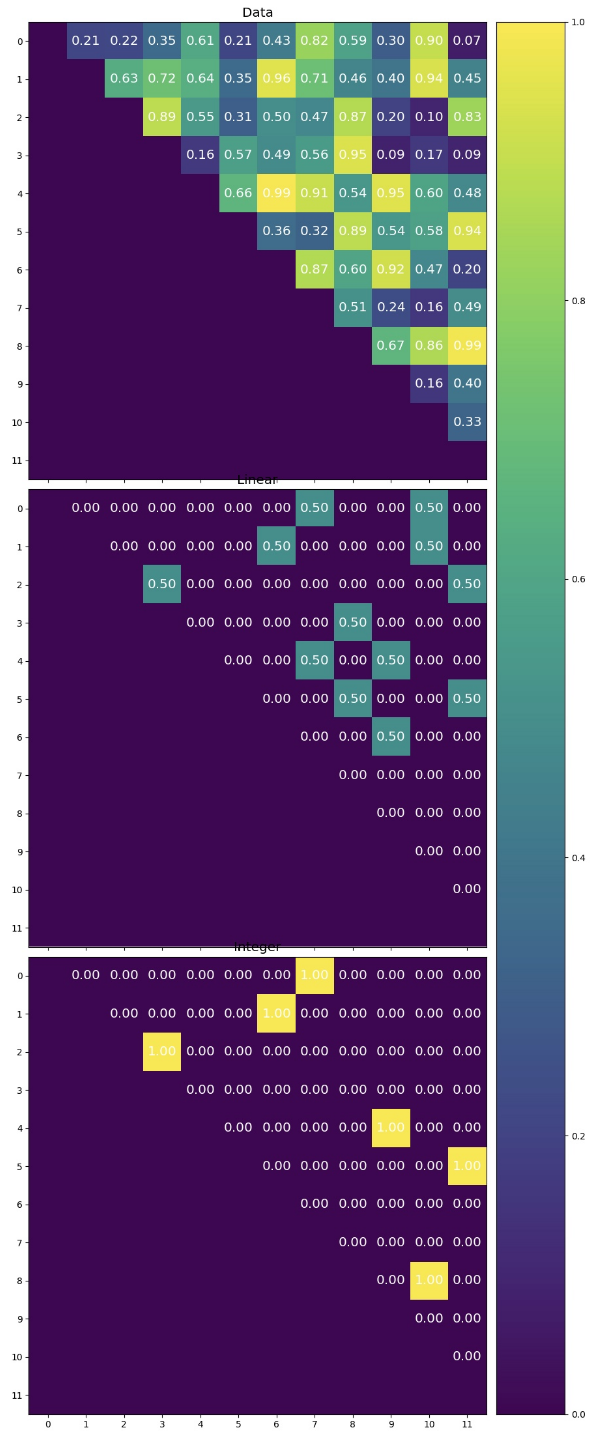
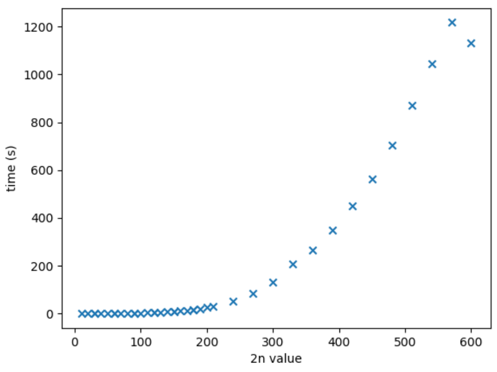


Figure 6: An illustrative example of how an LP relaxation may give non-integer solution when additional constraints are applied as in model 2. Top figure shows data matrix, middle figure shows the LP solution, and the below figure shows the IP solution.

An 2n = 12 example

In Figure 6, we show an illustrative example of how an LP relaxation may give non-integer solution in model 2. The linear solution all has 0.5 weights with the solution set $\mathcal{S} = (1\text{-}8, 1\text{-}11, 2\text{-}7, 2\text{-}11, 3\text{-}4, 3\text{-}12, 4\text{-}9, 5\text{-}8, 5\text{-}10, 6\text{-}9, 6\text{-}12, 7\text{-}10)$. This may look like arbitrary numbers, but when examining carefully, the solution set constructs a circle of 1-8-5-10-7-2-11-1, and a circle of 3-12-6-9-4-3. Other examples also showed this problem: non-integer solutions are formed by breaking integer solutions and form "large circles". The additional constraints prevent the model from forming circles of 3 vertices but larger circles are still formed. This result is as expected from the discussion in section $2n = 4$ Cases.

Computational cost and probability of integer solutions



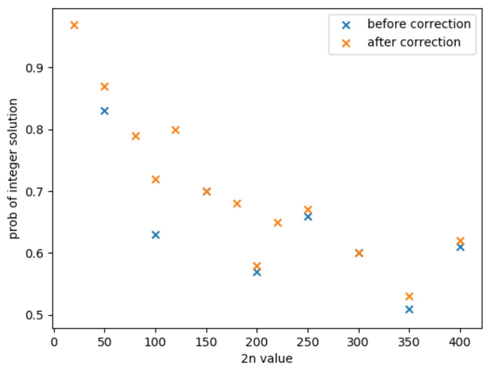


Figure 7: Computational cost and probability of having integer solutions. The left-hand side figure shows the computational cost of 1 trial, and the right-hand size figure shows the probability of having integer solutions on 100 trials.

In Figure 7 the computational cost when having 1 trial and probability of integer solutions when having 100 trials on the LP relaxation of model 2 is shown. The first figure shows the computational cost. We can see that the trend is similar as model 1 in Figure 4 but the computational cost is significantly increased since the figure showed here is just the computational cost of 1 trial (over 15 minutes for one trial when $2n$ is greater than 500). This is due to the increased amount of the total constraints.

The probability of integer solutions is shown in Figure 7 with datapoints collected before and after the bug fixing. We can see that after fixing the bug, the model performance in terms of giving an integer solution is improved. Although due to the computational cost, only partial data can be compared with Figure 5, we can see that the probability of having an integer solution is increased by the additional constraint. In model 1, the probability is less than 50% when $2n \geq 200$, but in model 2, the probability is still greater than 50% even when $2n = 400$. The additional constraints are effective on giving integer solutions.

4 Apply models to real-world data

After theoretically analyzing the effectiveness of LP relaxations of the maximum weight matching problem, we research how the proposed models could be used to solve real-world problems and how the real-world dataset would affect the performance of the proposed models.

Data acquirement

The raw data of infected patients numbers in different areas can be acquired at a regularly updated website built by Xinhuanet, an official state-run press agency in China [6]. We acquired two datasets via this website: 1) a coarse province-level dataset 2) a detailed city-level dataset. The coarse province-level dataset contains the infected patients numbers of 34 provinces including controversial areas. The detailed city-level dataset contains the infected patients numbers of roughly 500 cities. For the province-level dataset, I did not perform any pre-processing. For the city-level dataset, I manually cleaned the dataset by deleting unconfirmed areas, affected foreigners, and affected military units. After pre-processing, infected patients numbers of 412 cities are left and used to generate model input data matrix.

The acquired dataset is analyzed and visualized in Figure 8 and Figure 9. In Figure 8, we visualize the raw number of the infected patients in 34 provinces and 412 cities in the left-side figure and the right-side figure respectively. The y axis is the frequency of each number and the x axis is the logarithm of the number of infected patients. We notice that the data distribution is highly nonuniform due to the nature of infectious disease. In Figure 9, we visualize the data matrix (model input) that is generated via the raw dataset through a similar approach, where the y axis is the frequency of each matrix entry and the x axis is the logarithm of the number differences of infected patients between two areas. The province-level dataset has a relatively low frequency since the value $2n$ is too small. The city-level dataset has some high frequency datapoints although most of the datapoints have a relatively low frequency. Also, large differences tend to have lower frequency.

Unlike synthetic dataset, which contains unique numbers with a data range between 0-1, real-world dataset contains repeated numbers with a relatively wide data range. Also, it is worth noting that the distribution of infected patients has a "Long Tail Effect" since Covid-19 is an infectious disease. Before the disease becomes a pandemic, due to its infectious nature, only a small amount of cities have large affected patients numbers while most of

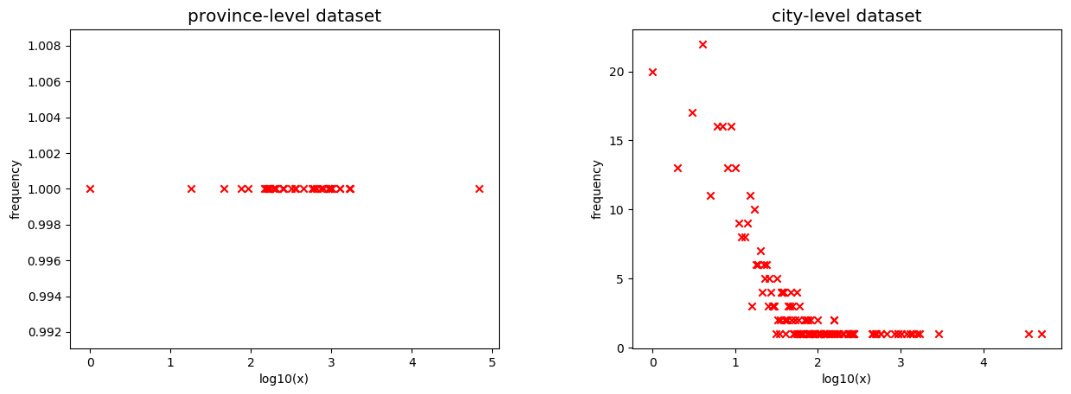


Figure 8: These two images are the overview of collected datapoints of diagnosed Covid-19 patients. The x axis is $log_{10}x$. Left-side figure is province-level dataset while the right-side figure is the city-level dataset.

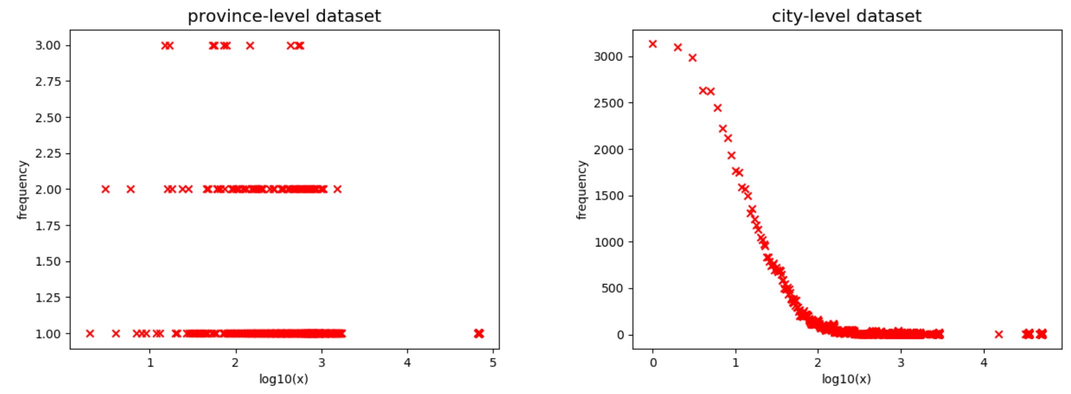


Figure 9: These two images are the overview of the differences of diagnosed Covid-19 patients. The x axis is $log_{10}x$. Left-side figure is province-level dataset while the right-side figure is city-level dataset.

Computational cost comparison				
Different models	pr-level real data	$n = 17$ sync data	city-level real data	$n = 206$ sync data
Model 1 linear	109 iters 0.00s	53 iters 0.00s	4563 iters 0.24s	278 iters 0.34s
Model 1 integer	109 iters 0.01s	53 iters 0.01s	30819 iters 1.30s	8993 iters 1.14s
Model 2 linear	1975 iters 0.05s	424 iters 0.02s	39754 iters 122.09s	2443 iters 305.02s
Model 2 integer	1975 iters 0.12s	424 iters 0.08s	1458 iters 241.58s	2443 iters 586.22s

Table 2: The computational cost comparison between real-world dataset and synthetic dataset with different $2n$ size on different models.

the cities only have a limited amount of affected patients. This unique nature of real-world Covid-19 dataset makes it distinguishable from synthetic datasets.

Computational cost

Table 2 shows the comparison of model performances between synthetic dataset and real-world dataset on different models. The province-level real-world dataset is compared with $2n = 34$ synthetic dataset; the city-level real-world dataset is compared with $2n = 412$ synthetic dataset.

For the province-level real-world dataset, the model gives similar performance comparing with $2n = 34$ synthetic dataset on the running time, where the synthetic dataset runs slightly faster. However, when considering the comparison between the city-level real-world dataset and the $2n = 412$ synthetic dataset, the real-world dataset runs significantly faster than synthetic dataset except for the integer version of model 1. This result may be caused by the difference of data distribution. When comparing the province-level real-world dataset with the synthetic dataset, since the total number of vertices is small enough, the data distribution of real-world dataset is similar with synthetic dataset since most of the data-points do not overlap. On the contrary, when considering the comparison between city-level real-world dataset with synthetic dataset, a lot of data-points are repeated and multiple optimal solutions may exist. Hence, the model may have a faster solving process comparing with synthetic dataset.

Note that different machines would give different computational time for the same model. For the $n = 206$ synthetic data experiments on model 2, I used a more powerful machine instead of my own laptop, which gives a faster solving time. If the code runs on my local laptop, it would require more time to run. For the following model challenging section, all experiments are conducted on the same machine (the faster one).

5 Challenge model

Aside from challenging model computationally, the probability of LP relaxation having integer solutions is also challenged in this section.

5.1 Challenge the model computationally

The IP and LP relaxation of Model 2 on $n = 206$ synthetic data are challenged. The original LP relaxation takes 305.02s to run; the original IP takes 586.22s to run (shown in Table 2).

Modifying model parameters (IP)

To improve computational performance of the model, I adjusted the model parameters including ”LazyConstraints”, ”PreDual”, ”Presolve”, ”MIPFocus”, ”Method”.

Lazy Constraints The parameter ”LazyConstraints” controls if the constraints are applied lazily. Intuitively, most of the constraints in both IP and LP will not be violated and an integer solution could be achieved at the first attempt. Since Gurobi does not support lazy constraints in any continuous models (LP, QP, or SOCP) naturally, I only test this on IP of model 2, where most of the constraints will never be violated. However, after setting the ”LazyConstraints” to 1 (enable lazy constraints), the model is significantly slower, which takes 10910.20 seconds. Instead of only changing the model parameter, I also attempted to write an optimization function to setup the lazy constraints manually. However, the model is still not improved.

PreDual The parameter ”PreDual” controls whether presolve forms the dual of the model. Changing this parameter to 0 (forbid forming the dual) cause the model to run significantly slower (885.78s), while setting it to 1 (forces the dual) improves the model (585.21s) and 2 (use a more expensive heuristic to decide) does not affect much (590.73s).

Presolve The parameter ”Presolve” controls the presolve level. When setting ”Presolve” as 0 and ”PreDual” as 0 at the same time, the parameter ”Presolve” benifit the program (from 885.78s to 732.17s). However, when the parameter ”PreDual” is not changed, setting ”Presolve” to 0 (do not presolve) causes the model to be slower (725.44s) and setting ”Presolve” to 1 (conservative presolve) makes the model 7% faster (546.10s).

Method The parameter ”Method” decides the algorithm used to solve continuous models. The specific methods selected and their corresponding computational cost are shown in Table 3. This change is the most effective one: the method of primal simplex, concurrent,

Computational cost comparison	
Method	Time
primal simplex	248.23s
dual simplex	684.62s
barrier	5389.66s
concurrent	269.27s
deterministic concurrent	584.61s
deterministic concurrent simplex	255.14s

Table 3: Training time via different methods (IP).

and deterministic concurrent simplex all reduce the computational cost for 50%. Among them, primal simplex is the most powerful one.

MIPFocus The parameter "MIPFocus" controls the MIP solver focus. Before changing this parameter, we set the method to primal simplex. Setting "MIPFocus" to 1 (focus on finding feasible solutions) barely affect the result (249.07s), and setting it to 2 (focus on proving optimality) and 3 (focus on the bound) will increase the time significantly (604.23s and 612.55s).

Modifying model parameters (LP)

I adjusted the model parameters including "PreDual", "Presolve", "Method".

PreDual After setting the "PreDual" to 0, which correspond to forbids presolve from forming the dual instead of using the default setting that uses a heuristic to decide, the model is speeded up to 225.70 seconds.

Presolve Setting "Presolve" to 0 (off) and 1 (conservative), respectively, will speed up the model to 209.56 seconds and 216.87 seconds.

Method The specific methods selected and their corresponding computational cost are shown in Table 4. This change is the most effective one again: the primal simplex method and the deterministic concurrent simplex method all reduced the time from more than 300 seconds to below 30 seconds. Similarly, the primal simplex is also the most powerful one.

Conclusion

Primal simplex and deterministic concurrent simplex are two powerful methods to solve this problem. These two methods greatly reduced the computational time required (58% for IP and 92% for LP), which also make the LP relaxation more advantageous comparing with the IP (23.40s comparing with 248.23s).

Computational cost comparison	
Method	Time
primal simplex	23.40s
dual simplex	448.08s
barrier	5002.09s
concurrent	316.55s
deterministic concurrent	345.61s
deterministic concurrent simplex	29.22s

Table 4: Training time via different methods (LP).

5.2 Challenge the probability of having integer solutions

The LP relaxation of Model 2 on $2n = 12$ and $2n = 20$ synthetic data of 10000 trials is challenged. The original probability is 0.9867 with computational time of 57.36s for $2n = 12$; and 0.9567 with computational time of 209.76s for $2n = 20$.

Adding additional constraints

Test 1: $\sum_{i=1}^{2n}\sum_{j=i+1}^{2n} x_{i,j} \geq n$. This constraint increases the probability of having integer solutions when the total amount of 0.5 solutions is odd (For example, when $n = 3$ and 2 solutions of 1 are replaced by 3 solutions of 0.5). For $2n = 12$, the probability is increased to 0.9884 with 64.83s. For $2n = 20$, the probability is the same but time is increased to 224.75s.

Test 2: $\sum_{h=1}^{i-1} x_{h,i} + \sum_{j=i+1}^{2n} x_{i,j} \geq 1$ for every i. These constraints actually have the same meaning with above Test 1 constraint but is slower.

Test 3: $x_{i,j} + x_{j,h} + x_{h,k} + x_{k,t} + x_{i,t} \leq 2$ for all distinct $i < j < h < k < t$. These constraints are originally designed to avoid a five-point-circle like i-j-h-k-t-i with a solution set $\mathcal{S} = (i\text{-}j, j\text{-}h, h\text{-}k, k\text{-}t, i\text{-}t)$. For $2n = 12$, the probability is increased to 0.9875 with 337.35s. For $2n = 20$, the probability is 0.9611 with 4445.09s. Under these constraints, the non-integer cases have the optimal solution of 1/3 and 2/3.

To conclude, test 1 and test 2 increased the probability only slightly. The effect of Test 3 is more obvious, but the computational time is increased greatly.

Setting random linear variables as integers

The original LP formulation requires all variables to be linear. Here we test if setting a certain percentage of them as integer helps. Results are shown in Table 5.

Probability of having integer solutions					
	linear	integer	50% integer	25% integer	10% integer
$2n = 12$	0.9867 57.36s	1.0 79.69s	1.0 98.44s	0.9995 90.81s	0.9963 94.99s
$2n = 20$	0.9567 209.76s	1.0 319.58s	0.9999 399.94s	0.9968 376.13s	0.9834 376.27s

Table 5: The probability of having integer solutions when setting random linear variables as integers. The trial size for both $2n = 12$ and $2n = 20$ is 10000.

When 50% of $x_{i,j}$ are set into integer, the probability is increased to 1.0 and 0.9999, respectively, for $2n = 12$ and $2n = 20$. When 25% of $x_{i,j}$ are set into integer, the probability is increased to 0.9995 and 0.9968, respectively, for $2n = 12$ and $2n = 20$. Even when only 10% of $x_{i,j}$ are set into integer, the probability is increased significantly to 0.9963 and 0.9834, respectively, for $2n = 12$ and $2n = 20$. This tells us that setting some of the linear variables as integer variables would help us get integer solutions effectively.

However, all of the computational time of above experiments increased because the original linear programming is changed to a mixed-integer programming (MIP). Also, since the program uses a probability method to determine which variable should be set to integer, the overall time increased even comparing to the IP (Tabel 5). This issue could be avoided if a better code is written. Here, we only show the possibility of having a better solution methodologically.

Overall, setting random linear variables as integers is a powerful method to increase the probability of having integer solutions.

References

[1] News article. "16 provinces to help Hubei cities fight coronavirus. Available in: `https://www.globaltimes.cn/content/1178997.shtml`". In: (2020).

[2] News article. "One-to-one help key to 16 cities in Hubei. Available in: `http://global.chinadaily.com.cn/a/202002/10/WS5e40975fa3101282172761e1.html`". In: (2020).

[3] Dae-Sik Choi and In-Chan Choi. "On the effectiveness of the linear programming relaxation of the 0-1 multi-commodity minimum cost network flow problem". In: *International Computing and Combinatorics Conference*. Springer. 2006, pp. 517–526.

[4] Ran Duan and Seth Pettie. "Linear-time approximation for maximum weight matching". In: *Journal of the ACM (JACM)* 61.1 (2014), pp. 1–23.

[5] Andreas Löbel. "Vehicle scheduling in public transit and Lagrangean pricing". In: *Management Science* 44.12-part-1 (1998), pp. 1637–1649.

[6] Data source. "Covid-19 situation report in China. Available in: `http://fms.news.cn/swf/2020_sjxw/2_1_xgyq/index.html?v=0.5339757488987171`". In: (2020).

Extra Materials

The manually collected dataset files can be found in "china.txt" and "china_detailed.txt". Logs for solutions that are manually recorded can be found in "6669_experiment_log.doc".