Matchings through Primal-dual algorithms

Chloé Daniel
LIG, UGA
Grenoble, France
chloe.daniel@etu.univ-grenoble-alpes.fr
Supervised by: Nguyen Kim Thang

I understand what plagiarism entails and I declare that this report is my own, original work.
Name, date and signature: Daniel,07/06/2024

Abstract

This paper explores matching algorithms in various settings using the Primal-dual method. Additionally, it examines how recent advancements in machine learning enhance the performance of some of these algorithms. We contribute by conducting experiments to observe the efficiency of incorporating machine learning based on different distributions.

1 Introduction

Optimization is the process of using tools and techniques to make critical decisions, obtain maximal efficiency and achieve desired outcomes. However, real-world problems are so complex, involving thousands of variables, that new methods are needed to be able to compute the desired optimal solutions. With the advent of machine learning, we explore optimization techniques that make use of this technology to assist us in the computation of those solutions.

Let's first consider a class of n students that has to choose an internship from n companies. All students, of course, have a different set of skills in every domain. The companies can hire only one student and want to hire the better suited student, given an affinity score between him and the company (1 being the best score and 5 the worst). This problem can be modelized as a graph, as shown in Figure 1

This problem is called minimum-cost bipartite matching. It's a combinatorial optimization problem studied extensively in a wide variety of settings (see [9][3] for examples). Combinatorial optimization is a well-studied field that has applications in every domain around us, from supply chain management to kidney donation programs. Knowing how to solve these problems correctly and efficiently is critical. For instance, failing to schedule properly in emergency response planning could have disastrous consequences.
Although the bipartite matching problem mentioned is known to have a polynomial time solution, it isn't the case for most combinatorial optimization problems. For this reason, it is necessary to relax the integrality constraint, stating that exactly one element should be matched to exactly one other. Once it is done, we look at what happens in the continuous domain. From there, a particular way of solving such problems has been widely generalized: Primal-Dual algorithms.

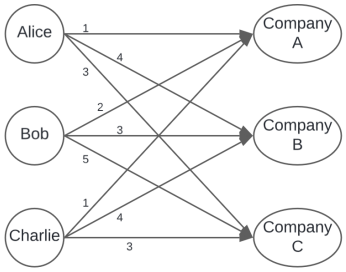

Figure 1: An instance of the matching problem of interns and companies

Looking back to our opening problem, we can model the relaxation as a linear program associating each student i to a company j by an affinity score a_{ij} given as input data.
The objective is to match each student to a company while minimizing the affinity score of this matching. For that, we create a matrix m where $m_{xy} = 1$ if the student is hired by the company and 0 otherwise.
Since the integrality constraint is relaxed, m_{xy} can take any value between 0 and 1: every student can divide their work time among several companies.

$$\begin{aligned} \min \quad & \sum_{i,j} a_{ij} \cdot m_{ij} & \\ \text{s.t.} \quad & \sum_{j} m_{ij} = 1, & \forall i \in \{0, \dots, n\} \\ & \sum_{i} m_{ij} = 1, & \forall j \in \{0, \dots, n\} \\ & m_{ij} \geq 0 & \forall i, j \in \{0, \dots, n\} \end{aligned}$$

$\sum_j m_{ij} = 1 \ \forall i$ states that every student should work for the exact amount of time dedicated to their internship. One student can work, for example, 60% of their time in a company and 40% in another, but no more, no less.

$\sum_i m_{ij} = 1$ is a similar constraint, but from the company perspective.
$m_{ij} \geq 0$ asserts that it isn't possible to work for a negative amount of time.

Duality is an operational research framework where we model the problem from two directions and name these the primal and the dual. Let's take a problem where the primal instance is minimizing the cost of production of a company. We can look at the dual problem, which is the maximization of the profit. In our situation, the dual program can be though of as finding the maximum satisfaction:

$$\begin{aligned} \max \quad & \sum_{i,j} p_i + q_j & \\ \text{s.t.} \quad & p_i + q_j \leq a_{ij}, & \forall i, j \\ & p_i, q_i \in \mathbb{R} & \forall i, j \end{aligned}$$

The Primal-dual technique builds a primal solution to the problem while maintaining a feasible dual one. This branch of algorithms are based on the weak and strong duality theorem.
Weak duality theorem : the value of any feasible solution of a primal minimization problem is always greater or equal to the value of any feasible solution of its dual maximization problem.

$$\sum_{i,j} a_{ij} \cdot m_{ij} \geq \sum_{i,j} p_i + q_j$$

Strong duality theorem : if the value of a primal solution equals the value of a dual solution, then both the primal and dual solutions are optimal.

$$\text{If } \sum_{i,j} a_{ij} \cdot m_{ij} = \sum_{i,j} p_i + q_j$$
$$\text{then } m_{ij}, p_i \text{ and } q_j \text{ are optimal solutions.}$$

In the survey of Buchbinder et al.[1], we can find several simple primal-dual algorithms of both online and offline settings. We focus here on the offline setting, and we generalize an algorithm [6] as follows [1]:

Begin with a feasible dual solution y.
At each iteration of the algorithm, we have a feasible dual solution y.

1. We construct a (possibly infeasible) primal solution x (determined by complementary slackness)
2. If x is a feasible solution, we stop
3. If x is infeasible, we find a new improved dual feasible solution y and go to Step 1.

Algorithm 1: Generalization of Primal-dual algorithms.

Once the continuous optimal solution is found, a discrete approximated solution can be computed by performing a rounding scheme.

In this paper, we explore existing bipartite matching algorithms in both online and offline settings by using the Primal-dual concept.
In the section 2, we focus on the waterfilling algorithm solving the online maximum unweighted bipartite matching. Using the Primal-dual tool to analyze the algorithm, we prove that the lower bound of the waterfilling algorithm is $(e-1)/e * OPT$ (OPT is the optimal solution), in other words, that our algorithm is $e/(e-1)$ competitive. We then observe our results in an experiment to further validate the competitive ratio.
In the section 3 of this paper, we explore a Primal-dual algorithm to solve the offline minimum weight bipartite matching. We present how machine learning can be used to enhance the performance of this algorithm, and we contribute by conducting a comparative performance experimental analysis. This experiment shows encouraging results, concurring with [2] that states that using machine learning can provide a speed-up depending on the distribution. These results motivates us to conduct further experiments to observe the performance of the machine-learning addition in different settings.

2 Online maximum unweighted bipartite matching

In this section, we study the online maximum unweighted bipartite matching problem. This problem has a wide variety of real-life applications where decisions have to be made without knowledge of future input such as Online Advertising or Ride-Hailing Services (Uber,Lyft,..).
To define this problem, let B=(V=$(L \cup R), E$) be a bipartite graph where L is the set of left vertices and R the set of right vertices such that $|L| = |R| = n$. E is the set of edges between L and R.
In the online setting, at initialization time, only L is known. At each step t=1,2...n one vertex of R is revealed along with its incident edges.
The goal of our algorithm is to maximize the number of selected edges at the end. The additional constraint is that when the algorithm makes a decision to select (or not) an edge (i, j) when j is revealed, the decision can not be reversed afterwards.
To make the problem more concrete, we can make a parallel with an ice cream vendor offering several flavors and wanting to maximize the quantity of ice cream he sold to his clients. Each flavor is represented by a vertex of L and every client by a vertex of R. However, he cannot give out more than one ice cream of each flavor. Each client likes a subset of the flavors. Once an ice cream is sold, he can not get it back, and he does not know the preference of his future clients. An edge between a client and a flavor signifies a preference of this client.
For this problem, computing the optimal solution is, in general, not possible. Since the best decision depends on future input, it is not possible to know if selecting an edge is optimal or not. In the ice-cream example, maybe the vendor decided to sell an ice-cream to a client, but realized later on that it would have been more profitable to sell him another flavor (a future client wanted only this specific flavor, and since it wasn't available anymore, he couldn't sell him anything). We

[1] The generalization of primal-dual algorithms has been inspired by the lecture notes of Alantha Newman [6]

thus want to get a worst-case approximation of the optimal solution.
We decide to study the linear relaxation of this problem, the fractional matching, as the integer problem is NP-complete.

$$\begin{aligned} \max \quad & \sum_{i,j} x_{ij} & \\ \text{s.t.} \quad & \sum_{j} x_{ij} \leq 1, & \forall i \in L \\ & \sum_{i} x_{ij} \leq 1, & \forall j \in R \\ & x_{ij} \geq 0 & \forall i \in L, j \in R \end{aligned}$$

The state-of-the-art algorithm for solving this problem is the waterfilling algorithm, as it provides the best competitive ratio $e/(e-1)$:
A competitive ration of $e/(e-1)$ means that, on any input, $\frac{e}{e-1} * ALG \geq OPT$.

The key concept of the waterfilling algorithm is to spend proportionally the fractional degree of every vertex in L: $d(i) = \sum_j x_{ij}$.
When a vertex j of R arrives and the set of neighbors of j, $N(j)$, is revealed, the algorithm chooses to select a certain quantity of every edge (i, j). To decide how much of an edge to select, the algorithm takes into account the previous fractional degrees of the i and assigns the highest fraction of the load to the edge (i, j) with the vertex i with the lowest fractional degree and the lowest fraction to the one with vertex i having the highest fractional degree. The intuition for these assignments is that a vertex with a high fractional degree is in high demand, and is thus more likely to be linked with another vertex in the future. When that happens, we naturally want to leave more opportunity for a future vertex to be linked with this "high-demand" vertex.
To get a more visual representation of this algorithm, we can imagine all the left vertices as being containers in which we can pour water. When j arrives, one unit of water must be distributed amongst its neighbors. The algorithm does it in such a way that it keeps the same water level for everyone. The algorithm is as follows:

At initialization $d(i) = 0 \; \forall i \in L$
When $j \in R$ arrives:

1. Compute $l(j)$ such that: $\sum_{i \in N(j)} max\{l(j), d(i)\} = 1 + \sum_{i \in N(j)} d(i)$
2. $l(j) = min\{l(j), 1\}$
3. $\forall (i, j) \in E$: $x_{ij} = max\{l(j), d(i)\} - d(i)$
4. Update $d(i)$ to $d(i) + x_{ij} \; \forall i \in N(j)$

$l(j)$ is the value that every $i \in N(j)$ will be set to if their current fractional degree is lower than this value. x_{ij} is the variable indicating the fraction with which vertex $i \in L$ is connected to vertex $j \in R$.

To prove that this algorithm has a competitive ratio of $e/(e-1)$, we use a Primal-dual method. Recalling the primal Linear Program, we obtain the following dual:

$$\begin{aligned} \min \quad & \sum_{i,j} p_i + q_j & \\ \text{s.t.} \quad & p_i + q_j \geq 1, & \forall (i, j) \in E \\ & p_i, q_j \geq 0 & \forall i \in L, j \in R \end{aligned}$$

Let $p_i = g(d(i))$ and $q_j = 1 - g(l(j))$
with $g(x) = \frac{e^x - 1}{e - 1}$

To prove that $p_i + q_j$ is feasible, we show that $p_i + q_j = g(d(i)) + 1 - g(l(j)) \geq 1$.

This is true, because $d(i) \geq l(j)$ (since step 3,4 of the algorithm) and thus:
$g(d(i)) + 1 - g(l(j)) \geq g(l(j)) + 1 - g(l(j)) = 1$ (g is non-decreasing).
Given that the solution is feasible, by weak duality:
$\sum_{i,j} p_i + q_j \geq \sum_{i,j} x_{ij}$
We claim[2] : $\frac{e}{e-1} \sum_{i,j} x_{ij} \geq \sum_{i,j} p_i + q_j$
Thus:
$\frac{e}{e-1} \sum_{i,j} x_{ij} \geq \sum_{i,j} x_{ij}$.

We can conclude that the waterfilling algorithm is $e/(e-1)$ competitive. [8] proves that this ratio is the best competitive ratio achievable for this problem.

Experiments

We now want to look at the experimental results of this algorithm and observe the distance between the theoretical bound and the approximation. For that, we implemented the online waterfilling algorithm [3] and compared it to the actual optimal solution. As an input, the online algorithm is given an instance of a graph where the right vertices and edges are revealed over time. The algorithm computing the optimal solution is given the same graph, but with all information already revealed. We observe (see Figure 2), that the approximation computed with the algorithm is, as expected, always between the optimal result(OPT) and the theoretical lower bound: $(e-1)/e * OPT$.

One natural extension of this problem would be to add weights to the edges and compute the maximum *weighted* bipartite online matching. However, there is no theoretical bound for this problem. See the following scenario:
We have a left vertex i and two right vertex j_1, j_2 arriving over time with their incident edges. At time t=1, j_1 reveals to be connected to i with a weight of 1. At time t=2, j_2 reveals to be connected to i with a weight of W arbitrarily large or small. With no knowledge of what will happen at t=2, the algorithm must make a decision at t=1. The algorithm could select the edge (i, j_1) and it would be the worst decision if W is large or the best if W is small. The reverse scenario happens if the algorithm decides to not select this edge. Therefore, with no

[2]The proof of this claim is not detailed here but can be found in the Cornell University lecture notes on Online bipartite matching algorithms[8].

[3]The code for this implementation is available here: https://github.com/chloe3701/Matching/tree/main/Online

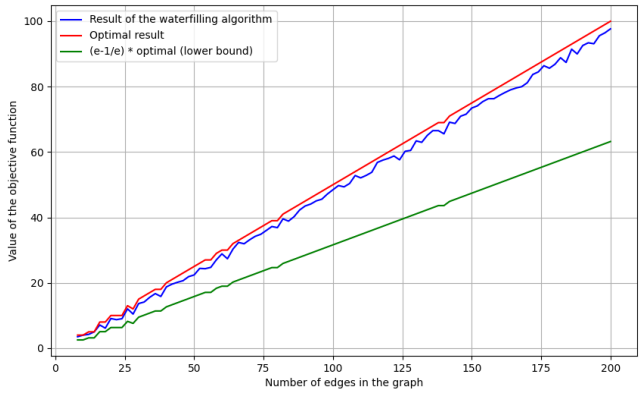

Figure 2: Comparison between the approximated and the optimal solution of the maximal online unweighted bipartite matching

prior knowledge of W, one can not judge the quality of the decision, since it is completely dependent on W. [4] obtains a bound by adding some assumptions to the problem.

3 Offline minimum weight bipartite matching

We now study the offline minimum weight bipartite matching, that was described in the introduction. Contrary to the online setting, in the offline setting, every information about the bipartite graph B=(V=($L \cup R$), E) is known at initialization time. We use a Primal-dual algorithm to build an optimal solution to this problem: we build a matching M (a subset of E), while maintaining feasible dual solutions $\vec{p}$ and $\vec{q}$.
To ensure that the algorithm will build a perfect matching, at each step we must have:

- $p_i + q_j \leq a_{ij}$: the dual solution is feasible.
- $\forall (i,j) \in M$ we should have $p_i + q_j = a_{ij}$. In other words, M should be a subset of tight edges. A tight edge is an edge satisfying the equality constraint.
- $|M|$ should always increase until it reaches n=$|L|$=$|R|$.

We represent our matching by a matrix m, where $m_{ij} = 1$, if $(i,j) \in M$, and 0 otherwise.
If all the constraints mentioned above are respected, then at termination time we will obtain:

$$a_{ij} = p_i + q_j \qquad \forall (i,j) \in M$$

And since $|M|$ = n we have:

$$m_{ij} = 1 \qquad \forall i$$
$$m_{ij} = 1 \qquad \forall j$$

Thus:

$$\sum_{i,j} a_{ij} \cdot m_{ij} = \sum_{i,j} p_i + q_j$$

By the strong duality theorem, the solution computed is the minimum perfect matching, since the solution is optimal.

Now, let's look at the algorithm that builds the matching M while maintaining the aforementioned constraints.

Algorithm:
Initialize M to $\emptyset$, and set $\vec{p} = \vec{q} = \vec{0}$.
Then, until $|M| = n$:
We look for an M-augmenting path P of tight edges, and if P exists, we augment the matching M with P by setting M to $M \oplus P$.
If such a path does not exist, then we need to adjust some dual variables to make additional edges tight.
When looking to adjust the dual variables, the simplest case would be to find a vertex i of L such that any edge containing i isn't tight. If that happens, we can simply raise p_i by some $\delta > 0$ until one of the edges containing i becomes tight.
However, it is possible that no such vertex exist. In this case, we need to increase the value of some p_i by some δ while lowering some q_j by the same amount without breaking the constraints.
To do that, we define a vertex set T (called Hungarian tree) that has the property that if one endpoint of an edge of M is in T then both are. We initialize this set to the set of free vertices (vertices that are not in M) of L.
We then repeat the following sequence of actions :

1. Compute $\delta = min\{a_{ij} - p_i - q_j | i \in L \cap T, j \in R \setminus T\}$, δ represent the maximum amount by which we can adjust the dual variables without breaking a constraint.
2. If $\delta > 0$, then we can set p_i to $p_i + \delta$ and q_j to $q_j - \delta$ for every i and j in the vertex set T. We call this the *dual adjustment* step.
3. If $\delta = 0$, then there exist at least one tight edge (i,j) between a vertex in T and a vertex outside T.
   From there, there are two possibilities: j is in the matching or j is not in the matching.
   - If it is, we can identify an edge (i',j) such that this edge belongs to the matching M. Then we add both vertices in T. We call this the tree growing step.
   - If it is not, then we can find an augmenting path P starting from a free vertex, going to i' and finishing in j. Using this path, we set M to $M \oplus P$ and finish this instance of Hungarian tree. This is the *augmentation* step.

[7] conducts a complexity analysis on this algorithm and find a complexity of $\mathcal{O}(mn^2)$.
However, it also shows that this complexity can be improved to $\mathcal{O}(mnlog(n))$ by using some bookkeeping techniques.

Enhancement of the algorithm with machine learning

Recent studies[2][5] have shown that it is possible to improve the performance of primal-dual algorithms using machine-learned predictions. Instead of starting the construction of the dual solution from the very beginning, one can make use of a machine-learned solution derived from a set of similar problems to obtain an optimal solution much faster. This is called a "warm-start". We look at [2] that provides us with a framework to build "warm-started" Primal-Dual algorithms using learned-based predictions.

This framework is build around the identification of three key challenges:

1. **Feasibility:** The learned predictions might be infeasible. It is thus necessary to design a function mapping the prediction to the set of feasible solutions in such a way that the mapped feasible solution is the closest to the original one.
2. **Optimization:** Once the learning-based prediction is feasible, ideally an algorithm can benefit from it to improve standard run-time guarantees.
3. **Learnability:** It is necessary to design a good prediction that can be learned in a relatively low number of instance. This solution should generalize well to similar problems.

Solving these three challenges gives us an algorithm that theoretically improves performance given the access to a relatively low number of similar problems.

We apply this framework to the algorithm described previously.
Let's first look at how to make our learned dual **feasible**.
For that, we compute r_{ij}, the "overflow" of each edge (i, j) of the graph. This number represents by how much the constraint $p_i + q_j \leq a_{ij}$ is violated.
We then need to find a vector $\vec{d}$ that solves:
$min \sum_{i \in V} d_i$
$st : d_i + d_j \geq r_{ij} \; \forall (i, j) \in E$
This is a variant of the set cover problem. We could solve this to get an optimal solution, but we are more interested in the speed efficiency of the computation, and therefore use a fast, greedy approach to get a 2-approximation of $\vec{d}$.

Initialize $\vec{d}$ to $\vec{0}$
While $\exists$ an edge in B:

- Select a vertex i of B
- While i has a neighbor
  - select j as the neighbor of i such that r_{ij} is maximum
  - $d_i = r_{ij}$
  - delete i and all incident edges in B
  - $i \leftarrow j$

return $\vec{d}$

Algorithm 2: 2-approximation algorithm computing the minimum amount by which the dual should be decreased to make it possible.

We then set $p_i \leftarrow p_i - d_i$ and $q_j \leftarrow p_j - d_j \; \forall i, j$
We end up with a feasible dual that is not too far from the learned dual predicted before. We name $\vec{y} = \vec{p} \cup \vec{d}$ the learned dual and $\vec{y}_{opt}$ the optimal dual. Assuming the distance between the learned dual and the optimal dual is $D = |\vec{y} - \vec{y}_{opt}|$, the distance between $\vec{y}_f$, the dual made feasible, and $\vec{y}_{opt}$ is at most $3 \cdot D$.

Optimization: From this feasible solution, we make it optimal by using the Primal-dual algorithm described previously. The only difference is that instead of initializing $\vec{p}$ and $\vec{q}$ to $\vec{0}$, we start from the dual variables that have been learned and made feasible. This is possible since given a feasible dual, the algorithm never increases it by an amount that would break the constraint.
Learning: To get an approximated dual solution, we fix the number of vertices of the graphs given as input, but the edge costs are drawn from an unknown distribution. We also associate each input graph to its optimal dual variable. Using these inputs, we use a linear regression to train the model.

Experimental observations

The main contribution of this paper is the implementation of the Primal-dual algorithm using "warm-started" dual and the comparison of its performance compared to the algorithm with a dual initialized to $\vec{0}$.
In all of our experiments, the bipartite graphs are generated such that the weights follow a given distribution, unknown by the algorithm.
We implemented the Primal-Dual algorithm with $\mathcal{O}(mn^2)$ complexity as well as the algorithm needed to train a model to predict a dual solution. We use a linear regression trained on 20 instances of problems. Finally we implement the approximation algorithm that takes a dual solution and makes it feasible.[4]. The result generated is the average of the results over 10 instances of graphs following the same distribution as the training data.
Our primary goal is to see the gain of the warm-start when the distribution of the graph varies.
The instances of the graphs follow a normal distribution with varying parameters. As a comparison metric, we introduce a third way to initialize the dual variables: the "mean dual". This dual is computed by taking the average of the optimal dual instances given in the training data set. Since a normal distribution is centered around a mean, we wanted to look if starting from "mean" dual variables, would yield a satisfying speedup. The dual in the classic Primal-dual algorithm is always set to $\vec{0}$ and the learned Primal-dual start with a predicted dual. Our goal is to provide an in-between to verify if using machine-learning is really much better or if starting from an approximated non-zero solution would be enough.
We first look at the performance of the algorithm when setting the normal distribution to $\mathcal{N}(100, 10^2)$. We observe in Figure 3 that with an instance of a graph that is large enough, using learned dual is more efficient. While the "mean" dual show a slightly better performance, it is negligible compared to the learned dual.
However, when increasing the standard deviation, the performance of the algorithm learned dual decreases (as shown in Figure 4). Its performance becomes similar to the classic algorithm and sometimes performs worse. This is due to the fact that machine learning generalizes well the learned data when it is consistent. If the variance is high, it is unable to gain information from the data given.

[4]The code for this implementation is available here: https://github.com/chloe3701/Matching/tree/main/Learning_augmented

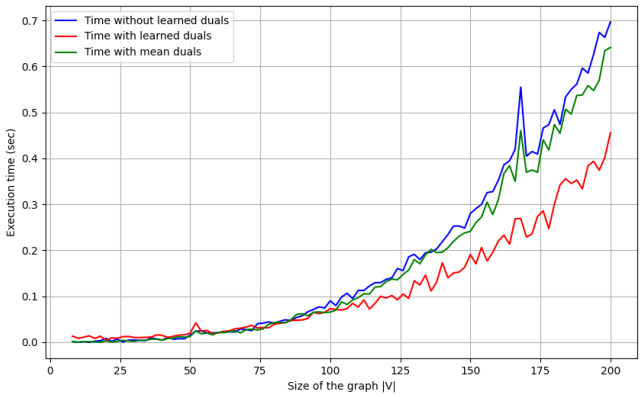

Figure 3: Performance of the Primal-dual solution with and without "warm-started" dual. The distribution follows $\mathcal{N}(100, 10^2)$

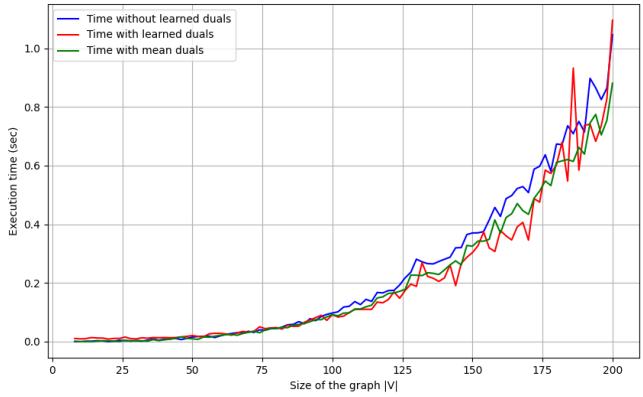

Figure 4: Performance of the Primal-dual solution with and without "warm-started" dual. The distribution follows $\mathcal{N}(100, 100^2)$

This results can prove to be very useful as when applied in real life, an algorithm tends to be used on data that is very similar.

Future experiments

These results are encouraging, but more experiments should be conducted:

1. We first need to improve the algorithm used in the implementation to one with lower time complexity $\mathcal{O}(mnlog(n))$. It is possible that the speedup observed would become less significant with an algorithm with a lower complexity. It would also allow us to conduct experiments on larger graph instances.
2. Experiments on different kind of variances should be conducted. We need to know if the speedup observed holds on different distribution or if is limited to normal distribution with relatively low variance. This is useful to determine in which scenario making use of "warm-started" dual becomes interesting.
3. Finally, we should experiment on using trained models on graphs of various sizes. At the moment, the model trained are used on graph with the exact same size. However, it often happens in real life that the data fluctuates slightly in size. One very performant model would become useless if it could not be reused on a graph with size that varies a little.

4 Conclusion

In this paper, we explored the application of Primal-Dual algorithms in solving both online and offline bipartite matching problems. We started with the online maximum unweighted bipartite matching problem, and studied the waterfilling algorithm. Through theoretical analysis and experimental validation, we demonstrated that the waterfilling algorithm achieves a competitive ratio of $\frac{e}{e-1}$.

We then looked at the offline setting, focusing on the minimum weight bipartite matching problem. Utilizing a Primal-Dual approach, we worked through an algorithm that constructs an optimal matching while maintaining feasible dual solutions.
Furthermore, we delved into the integration of machine learning to enhance the performance of Primal-Dual algorithms by using machine-learned predictions to provide a "warm start" for the algorithm's dual.
Utilizing a provided framework, we implemented it to conduct comparative performance experiments.
Our result showed a promising potential in using machine learning in the resolution of this problem by showing a noticeable speedup with a distribution with reasonable variance.

Further work is needed to validate those findings: improving the time complexity of the algorithm used, conducting more experiment on different distribution and varying graph sizes.

Overall, we studied online and offline settings for matching problems extensively. One more interesting venture to explore would be the semi-streaming setting that situates itself in between online and offline setting.

References

[1] N. Buchbinder and J. S. Naor. The design of competitive online algorithms via a primal–dual approach. *Foundations and Trends® in Theoretical Computer Science*, 3(2–3):93–263, 2009.

[2] M. Dinitz, S. Im, T. Lavastida, B. Moseley, and S. Vassilvitskii. Faster matchings via learned duals. *CoRR*, abs/2107.09770, 2021.

[3] Y. Emek, Y. Shapiro, and Y. Wang. Minimum cost perfect matching with delays for two sources. *Theoretical Computer Science*, 754:122–129, 2019. Algorithms and Complexity.

[4] T. Kesselheim, K. Radke, A. Abels, and B. Vöcking. An optimal online algorithm for weighted bipartite matching and extensions to combinatorial auctions. 09 2013.

[5] E. Kevi and N. K. Thang. Primal-dual algorithms with predictions for online bounded allocation and ad-auctions problems, 2024.

[6] A. Newman. Lecture notes in optimization and approximation, November 2016.

[7] C. University. Cs 6820: Algorithms lecture notes: Primal-dual min-cost bipartite matching. Lecture notes, 2010. Fall 2010 Lecture.

[8] C. University. Cs 6820: Algorithms lecture notes: Online bipartite matching algorithms. Lecture notes, 2012. Fall 2012 Lecture.

[9] T. Öncan and Kuban Altınel. A branch-and-bound algorithm for the minimum cost bipartite perfect matching problem with conflict pair constraints. *Electronic Notes in Discrete Mathematics*, 64:5–14, 2018. 8th International Network Optimization Conference - INOC 2017.