A Study of Recent Linear Programming-Based Algorithms for the Maximum Weight Bipartite Matching Problem

Research Report

Mahdi Qaempanah*, Mohammad Amin Raeisi†
*Department of Computer Engineering, Sharif University of Technology
†Department of Mathematical Sciences, Sharif University of Technology

***Abstract*—Following Kuhn's breakthrough work on the Hungarian Algorithm, the graph-matching problem has been extensively studied across various settings. Recently, Zheng and Henzinger proposed an LP-based solution to find a $(1-\epsilon)$-approximate maximum weighted matching in bipartite graphs. Although their initial algorithm terminates in $O(m\epsilon^{-1}\log(\epsilon^{-1}))$ time, they improved it to $O(m\epsilon^{-1})$, surpassing all previous approximate algorithms for the problem. We conducted a study to understand this algorithm and related optimization techniques. Furthermore, we implemented and analyzed this algorithm, along with the Hungarian algorithm, the direct reduction to LP algorithm, and the path-growing algorithm, comparing their performance. To our knowledge, this represents the first computational study of matching algorithms employing optimization techniques to address this problem.**

***Index Terms*—Maximum Weight Bipartite Matching, Linear Programming, Packing, Multiplicative Weight Update, Computational Study**

I. Introduction

Graph matching is one of the most well-studied problems in combinatorial optimization and has many applications in job scheduling, transportation, network problems, and computer science areas. Consider a weighted bipartite graph $G = (V, E)$, where each edge $e = (u, v)$ is assigned a non-negative weight w_e. The objective of the maximum weight bipartite matching problem is to identify a subset of edges $M \subseteq E$ such that no two edges e and e' in M share a common endpoint, and the total weight of this subset is maximized—formally, to maximize $\sum_{e\in M} w_e$ among all feasible sets M. To address this problem, one may formulate the following relaxed linear programming program:

$$\begin{aligned} \text{maximize} \quad & \sum_{e\in E} w_e x_e \\ \text{subject to} \quad & \sum_{e\in N(v)} x_e \leqslant 1, \qquad v \in V \\ & x_e \geqslant 0, \qquad e \in E \end{aligned} \tag{1}$$

The dual program of (1) is as follows:

$$\begin{aligned} \text{minimize} \quad & \sum_{v\in V} y_v \\ \text{subject to} \quad & y_u + y_v \geqslant w_e, \qquad e = uv \in E \\ & y_v \geqslant 0, \qquad v \in V \end{aligned} \tag{2}$$

It has been proven that the (1) linear programming formulation has integral optimal solutions, thereby resolving the problem when solved. Nevertheless, as the LP problem is more general than the MWM problem, these algorithms result in complex implementations and higher time complexities. An alternative approach to address this challenge involves recognizing that the given LP falls under the category of Packing LP problems. Recent advancements in solving Packing LP problems, such as those achieved through multiplicative weight updates, are highlighted in works like [1], [2]. While the problem can be solved polynomially, the high time complexity of exact algorithms has prompted exploration in approximate settings. Recently, Zheng and Henzigner [3] introduced a multiplicative auction algorithm, presenting a Fully Polynomial-Time Approximation Scheme (FPTAS) for the problem. Their solution outperforms all previous approximate maximum weighted bipartite matching, achieving a $O(m\epsilon^{-1})$ complexity. Using the formulation of (1) and its dual problem, their algorithm builds upon recent results in optimization. Prior to their work, Duan and Pettie proposed a more intricate LP-based algorithm for general graphs [4]. We have delved into advanced Optimization topics employed in these algorithms and examined their practical effectiveness. Additionally, we have compared the results of LP-based algorithms with non-LP-based algorithms. This comparative analysis seeks to ascertain whether optimization tools confer advantages in addressing this crucial computer science problem.

A. Related Work

The work on the matching problem was initiated by Kuhn in [5], who proposed the Hungarian algorithm. The Hungarian algorithm is a straightforward approach that operates in $O(n^5)$. Subsequently, numerous researchers introduced improved exact algorithms to address this problem. Recently, Chen et al. presented an $O(m^{1+o(1)})$ algorithm in [6] to solve the max flow, which is a more general version of this problem.

Consideration of a more intricate version of this problem arises in dynamic settings, where the edges and vertices of the graph can change. The latest result in this setting is attributed to Zheng and Henzinger, who presented an $O(m\epsilon^{-1})$ algorithm in [3] to tackle the bipartite version of the problem, allowing for the deletion of vertices on one side and the addition of vertices to the other side of the graph. Their algorithm is based on multiplicative weight updates, extensively studied in recent works, such as [1], [2], [7], to solve more general versions of the Maximum Weight Matching

(MWM) problem, known as Packing-LP problems.

This problem has also been explored in various computational settings, including online and streaming models [8], [9]. This shows the importance of the problem in the area of computer science and makes this computational study more valuable.

B. Our Contribution

In this report, we introduce the multiplicative auction algorithm [3], along with two exact algorithms and an additional approximate algorithm. We then analyze the practical performance of these algorithms by comparing their execution times, outputs, results, and the approximation ratio of the approximate algorithms. To do so, we generate various tests and execute the algorithms on these tests. Our analysis addresses the effectiveness of the multiplicative auction algorithm in practical scenarios and its convergence speed to exact solutions for different ϵ values. Our implementations of these algorithms, along with our test generators and output receivers, are available in our GitHub Repository.

C. Outline

The subsequent sections of this paper are structured as follows: In Section II, we explain three previous algorithms for the problem, which we have implemented for comparison with the recent LP-based algorithm. Section III delves into the multiplicative auction algorithm, and introduces the optimization techniques used. Our experiments and the obtained results are presented in Section IV, while Section V provides a brief conclusion and highlights open questions based on our experiments.

II. Previous Algorithms

In this section, we introduce several algorithms that are relevant to our main algorithm. We have implemented these algorithms and will conduct a practical analysis and comparison to the auction-based algorithm in the III section.

A. Direct Reduction to LP

It is proved [10] that the program described in (1) always has an integral solution. It is evident that the integral solutions of this program correspond to matchings in the graph. Therefore, the optimal solution of (1) not only serves as an upper bound for the maximum matching problem but also provides an exact solution. Consequently, one can easily obtain the correct answer by solving the program using well-known LP solvers such as the interior point method. The time complexity of this solution is the same as the complexity of solving an LP with m variables and n constraints over these variables.

B. Path Growing Algorithm

In [11], Drake and Hougardy proposed a simple greedy algorithm with a $\frac{1}{2}$ approximation ratio for the general maximum matching problem. In contrast to the Direct Reduction and Hungarian algorithms, this algorithm is neither based on linear programming techniques nor an exact algorithm. It iteratively selects a non-deleted vertex and extends a path from this vertex by choosing the heaviest edge incident to the current endpoint of the path and deleting the endpoint vertex. It alternately assigns edges to two sets M_1 and M_2 and finally outputs the set of edges with maximum weight among the sets M_1 and M_2. The pseudocode illustrating the algorithm is provided in Algorithm 1.

Algorithm 1 Path Growing Algorithm ($G = (V, E), w : E \mapsto \mathbb{R}_+$)

```
M_1 ← ∅, M_2 ← ∅, i ← 1
while E ≠ ∅ do
    Choose x ∈ V of degree at least one arbitrarily
    while x has a neighbor do
        Let xy be the heaviest edge incident to x
        Add xy to M_i
        i ← 3 − i
        Remove x from G
        x ← y
    end while
end while
return max(w(M_1), w(M_2))
```

Lemma 1: The path-growing algorithm terminates in $O(m)$.

Proof: In the inner loop of the algorithm, we never encounter a duplicated value x because once we process a vertex in this loop, it gets deleted. Additionally, the time complexity of the inner loop for a vertex x is $O(\deg_x)$. Therefore, the total time complexity is $O(m)$. ∎

Lemma 2: The path-growing algorithm has a $\frac{1}{2}$ approximation factor.

Proof: Once a vertex is deleted, we assign all its remaining incident edges to this vertex. Therefore, each edge is assigned to exactly one vertex. Now, let's consider a maximum matching M^* in G. Since M^* is a matching, all of its edges are assigned to different vertices. As we pick the heaviest remaining edge incident to the vertex in each step of the algorithm, it follows that $w(M_1 \cup M_2) \geqslant w(M^*)$. Consequently, $max(w(M_1), w(M_2)) \geqslant \frac{1}{2} w(M^*)$. ∎

C. Hungarian Algorithm

The Hungarian method was initially proposed by Kuhn in [5]. Initially, the problem is transformed into finding the maximum perfect matching in a complete bipartite graph by introducing dummy vertices and edges. This conversion results in the first set of constraints in the matching LP (1) becoming equalities, and the nonnegativity constraints in the dual matching LP (2) being eliminated. The main idea is then to find a matching and a set of y values satisfying the complementary slackness conditions, leading to an optimal solution. Before describing the algorithm, let us introduce some important definitions.

Definition 1 (Tight Edge): For a dual solution y, an edge $e = uv$ is tight if and only if $y_u + y_v = w_e$.

The definition of tight edges helps us find a combinatorial interpretation for the complementary slackness conditions:

Lemma 3: Given a feasible solution y for the dual program, there exists a perfect matching M in G such that M and y are corresponding optimal solutions to primal and dual programs if and only if there is a perfect matching on the tight edges of G.

The algorithm operates according to the steps outlined in Algorithm 2. The ϵ value in lines 11 and 12 denotes the maximum value such that the new y values remain feasible. The sets referenced in the algorithm are visually represented in Figure II-C, aiding in the comprehension of their roles and interactions within the Hungarian Algorithm. A naive implementation of this idea leads to an $O(n^5)$ algorithm. However, through some optimizations, we reduced the time complexity to $O(n^3 \log n)$ in our implementations.

Algorithm 2 Hungarian Algorithm ($G = (A \cup B, E)$ is a complete bipartite graph where $|A| = |B|$, $w : E \mapsto \mathbb{R}_+$)

1: $y_v \leftarrow 0$ for every $v \in B$
2: $y_v \leftarrow max_{u \in N(v)}$ for every $v \in A$
3: **while** there is no perfect matching on tight edges **do**
4: $M \leftarrow$ a maximum matching on tight edges
5: Orient M edges from B to A
6: Orient edges not in M from A to B
7: $S \leftarrow$ the free (not incident to M) vertices of A
8: $T \leftarrow$ reachable vertices from S
9: $O \leftarrow T \cap$ {non-free vertices of B}
10: $O' \leftarrow$ the other endpoints of vertices in O in M
11: Set $y_u \leftarrow y_u + \epsilon$ for every $u \in O$
12: Set $y_u \leftarrow y_u - \epsilon$ for every $u \in S \cup O'$
13: **end while**
14: **return** a perfect matching on tight edges

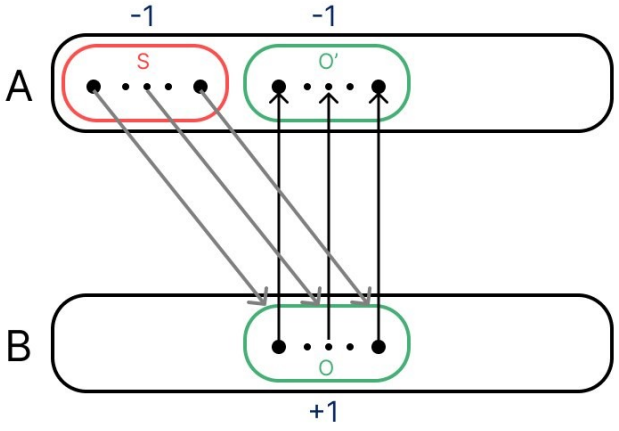

Fig. 1. Referenced Sets in the Hungarian Algorithm

III. Multiplicative Auction Algorithm

In this section, we explain the multiplicative auction algorithm for maximum weighted bipartite matching [3]. The primary focus of this project is to analyze this algorithm and compare it with other algorithms. Therefore, we first need to have an understanding of what this algorithm does.

The auction algorithms, proposed by Bertsekas [12] and separately by Demange, Gale, and Sotomayor [13], are closely related to primal-dual solutions. This means that they iteratively improve the primal solution (i.e., the matching) and the dual values (denoted as y values in our problem) until they reach some suboptimal state. In the auction algorithms, we interpret the dual variables as prices, and we have two sets of buyers and goods aiming to reach some optimal assignment.

Suppose the input bipartite graph has two parts U and V, where U represents a set of goods and V represents a set of buyers who are seeking to purchase items from U. We are looking for an assignment of goods to buyers such that each buyer gets at most one good and each good is assigned to at most one buyer, aiming for a $(1-\epsilon)$-approximation of maximum assignment utility.

First, the algorithm scales all the weights so that all the weight values become $O(\frac{n}{\epsilon})$. It can be shown that since we only want a $(1-\epsilon)$-approximation, this is possible. Then, the algorithm initializes all the prices with 0 (starts with $y_u = 0$ for every $u \in U$), and the matching is empty (no buyer has bought any items). We define the utility of u for the buyer v, or $util(uv)$, as follows:

$$util(uv) = w(uv) - y_u \tag{3}$$

Then the algorithm iteratively lets some buyer v, who is not covered by the matching, buy the item with maximum utility for v. We call this item u_v. If $util(u_v v) < 0$, the buyer does not buy any items. Otherwise, the algorithm matches u_v to v and increases the cost of y_{u_v} to $y_{u_v} + \epsilon \cdot w(u_v v)$. Consider that there might be some other buyers v' which are previously matched to u_v. In that case, we repeat the same procedure recursively with v' and try to match it with some goods. The detailed pseudocode of this algorithm is illustrated in Algorithm 3. In this algorithm, we utilize an internal function called MatchR, which is responsible for matching a buyer to the item with maximum utility. This function plays a crucial role in the overall process and facilitates the assignment of goods to buyers based on their utility. For detailed implementation, refer to Algorithm 4.

It can be proved that this scheme outputs a $(1-\epsilon)$ solution for the problem. To establish this, we introduce a relaxed version of the complementary slackness lemma as follows.

Lemma 4: Let M be a matching and y be the corresponding dual (price) values. If the following conditions hold:

1) For every $uv \in E$, $y_u + y_v \geqslant (1-\epsilon_0) \cdot w(uv)$,
2) For every $e \in M$, $y_u + y_v \leqslant (1+\epsilon_1) \cdot w(uv)$,
3) For every unmatched vertex u, $y_u = 0$.

Then M is a $\big((1+\epsilon_1)^{-1}(1-\epsilon_0)\big)$-approximate solution for the maximum weighted matching.

This lemma is very useful in the maximum matching problem literature and was first utilized by Duan and Pettie in [4] for their scaling algorithms. The proof operates by establishing a relationship between the weight of the current matching M

Algorithm 3 Multiplicative Auction Algorithm ($G = (U \cup V, E), w : E \mapsto \mathbb{R}_+$)

Initialize $M \leftarrow \varnothing$
Initialize $y_u \leftarrow 0$ for every $u \in U$
$Q_v \leftarrow \varnothing$ for every $v \in V$
$L \leftarrow \varnothing$
for j from ϵ to 1 in multiples of $\frac{\epsilon}{2}$ **do**
    $i \leftarrow \lfloor log_{1+\epsilon}(j \cdot w(uv)) \rfloor$
    Add (i, uv) to L
end for
Sort L in decreasing order with radix sort
for (i, uv) in L **do**
    Add (i, uv) to the back of Q_v
end for
for $v \in V$ **do**
    MatchR(v)
end for

Algorithm 4 MatchR(v)

while Q_v is not empty **do**
    $(j, uv) \leftarrow$ the first element of Q_v
    $j_v \leftarrow j$
    $util(uv) \leftarrow w(uv) - y_u$
    if $util(uv) < (1+\epsilon)^j$ **then**
        Remove (j, uv) from Q_v
    else
        $y_v \leftarrow util(uv)$
        $y_u \leftarrow y_u + \epsilon \cdot w(uv)$
        if u was matched to v' so that $uv' \in M$ **then**
            Remove (u, v') from M
            $y_{v'} \leftarrow 0$
            Add (u, v) to M
            MatchR(v')
            return
        else
            Add (u, v) to M
            return
        end if
    end if
end while

and the y values, and subsequently linking the solutions of the dual program (2) to the weight of the optimal matching M^*. This approach allows for a clear understanding of how the y values influence the weight of the matching and ultimately lead to the approximate optimality of M.

Using lemma 4 and selecting suitable values for ϵ_0 and ϵ_1, it can be demonstrated that the conditions necessary for achieving a $(1-\epsilon)$-approximate algorithm are satisfied in the solution of the algorithm. Furthermore, this algorithm runs in $O(m\epsilon^{-1})$. Hence, this FPTAS outperforms the previous approximate matching schemes for the weighted bipartite matching problem theoretically.

IV. Experiments

A. Setup and Implementations

The implementation of all algorithms, test generators, and the results of algorithm executions are all available on our GitHub Repository. All the algorithms mentioned below are implemented in Python and executed on a computer equipped with an AMD Ryzen 7 4800H processor and 16GB of RAM. The reported times are in milliseconds.

In the following sections, we report the results obtained from the execution of the following algorithms:

- Multiplicative Auction Algorithm
- Hungarian Algorithm
- Direct Reduction to LP Algorithm
- Path Growing Algorithm

We utilized the CVXPY library for solving the linear programming problem involved in the `direct reduction` approach. For the `path growing` algorithm, instead of repeatedly searching for new, undeleted vertices, we adopted a simpler iteration strategy, iterating over all vertices. Basic data structures like lists were employed for implementation. In implementing the `multiplicative auction` algorithm, we chose a non-recursive approach for Algorithm 3 to prevent excessive recursion depth, which could lead to increased cache memory usage. Lastly, in the `Hungarian` algorithm, we optimized the process by finding a maximum matching only in specific rounds rather than every iteration. Additionally, for each vertex, non-tight edges were kept in sorted order every iteration, resulting in an efficient $O(n^3 \log n)$ implementation.

B. Generators and Testbed

The tests used for analyzing the algorithms and their results were obtained through two methods:

- We derived a few of our test cases from well-known datasets and real-world graphs, many of which exhibited non-random structures. While some were already bipartite graphs, others were directed graphs that we split to create bipartite graphs. Since most real-world graphs are not naturally bipartite, curating these tests presented a significant challenge. We collected these graphs primarily from the Stanford dataset and Renchi dataset. It is worth noting that due to the problem's relevance in various applications, studying and analyzing these tests holds great importance. Our test suite includes a total of 5 test cases in this category.
- Several test cases have been created through various generators that we implemented. To generate these tests, we classified the test cases in three ways:
    1) We divided the tests into two categories, cardinal (*c*) and weighted (*w*). The weights in cardinal graphs are 1 and in the weighted graphs, edges have positive weights.
    2) For the weighted graphs, we divided them into low-weight graphs (*l*) and high-weight graphs (*h*). The weights in low-weight graphs are in the range

$[1, 100]$ while in high-weight graphs, the weights are in the range $[1, 100000]$.

3) We divided graphs based on their structures into sparse (*s*) and dense (*d*) graphs. In sparse graphs, $m \in O(n)$ while in dense graphs $m \in O(n\sqrt{n})$.

Finally, for each combination of these categories, we generated 15 tests. For instance, the *wdh* test group represents the tests of weighted dense graphs with high weights. In 5 of these tests, n is increased gradually approximately from 10 to 10000, and in 10 other tests, the n values are around 100.

C. Analysis

1) General Results and Outputs: Results are available in full on our GitHub Repository. Here, for analysis, outputs for a selected set of tests are shown in Table IV-C1. Furthermore, approximation ratios of the `multiplicative auction` and `path growing` algorithms are presented in Table IV-C1. The `multiplicative auction` is executed with $\epsilon = \frac{1}{2}$ for comparison with the `path growing` algorithm in terms of their approximation ratios. Furthermore, the execution of every test is halted after 20 seconds if it does not produce any visible progress within this interval. In such cases, the absence of output is indicated by '-'.

test	$\|V\|$	$\|E\|$	reduction	Hungarian	multiplicative	path growing
cs_ct1	385	644	177	177	174	137
cd_ct3	352	12052	164	164	164	162
wsl_t4	1595	916	-	27997	27500	18376
wsh_ct9	320	17229	15568264	15568264	13306701	15291471
wdl_ct7	308	133	4179	4179	4072	2491
wdh_ct8	280	7189	12000426	12000426	10854309	11709158
DBLP	7525	29257	-	-	4313	3303

TABLE I
OUTPUTS AND OVERALL RESULTS

test	multiplicative	path growing
cs_t4	0.95	0.75
cd_ct9	1	0.99
wsl_ct3	0.97	0.70
wsh_ct3	0.90	0.97
wdl_t4	0.96	0.67
wdh_ct3	0.87	0.99

TABLE II
APPROXIMATION RATIOS FOR NON-EXACT ALGORITHMS

The first observation that can naturally be made is that the output values of the `Hungarian` and `direct reduction` algorithms are always equal. This is because both algorithms are exact, and their output is the optimal solution. Additionally, it can be observed that the approximation ratio in both the `multiplicative auction` and `path growing` algorithms is always greater than or equal to 0.5, as expected. However, it can be seen that the approximation ratio of the `multiplicative auction` algorithm is usually not only better than the `path growing` algorithm but also within a small margin of 1. It suggests that the `multiplicative auction` algorithm is very effective in practice even for larger values of ϵ. This raises a question of how quickly the answer of the `multiplicative auction` algorithm converges to the optimal solution based on ϵ. We will further investigate this question in the IV-C3 section.

2) Execution Time: The execution times for all tests are available on our GitHub Repository. Here, we present the execution times for a selected set of tests in Table IV-C2 for analysis purposes.

First, observe that the execution time of the `Hungarian` and `reduction` algorithms is significantly higher than the execution time of the approximation algorithms. For instance, the execution time of the `Hungarian` and `reduction` algorithms on the `cd_ct5` test is approximately around 9.1 and 5.2 seconds, respectively, while for the approximation algorithms, it is less than 0.1 seconds. This disparity arises due to the time complexity of the `Hungarian` algorithm being $O(n^3 \log n)$ and the `reduction` algorithm solving a linear program with m variables and n constraints, resulting in a very large time complexity. Typically, the `reduction` algorithm exhibits better performance in terms of execution time, but in certain cases, such as the `wdh_ct2` test, the `Hungarian` algorithm produces output more quickly.

Since both approximate algorithms operate in approximately $O(m)$ time, they provide very quick responses, even in larger and denser tests such as the `WikiVote` test, where both algorithms terminate in less than 1 second. Furthermore, it can be observed that the `path growing` algorithm generally outperforms the `multiplicative auction` algorithm. This difference in performance can be attributed to the larger constant factor in the `multiplicative auction` algorithm, stemming from its more complex implementation and the ϵ^{-1} constant.

test	$\|V\|$	$\|E\|$	reduction	Hungarian	multiplicative	path growing
cs_ct1	385	644	366.39	14035.65	8.40	1.13
cd_ct5	278	11993	9156.07	5241.19	93.54	5.71
wsl_ct4	349	346	257.99	12550.08	4.69	0.37
wsh_ct7	302	7218	3858.85	7938.36	26.99	3.49
wdl_t4	1581	1395	1159.35	-	16.83	3.65
wdh_ct2	368	25809	-	12557.61	113.31	26.55
WikiVote	16573	103690	-	-	654.86	111.67

TABLE III
RUNTIME RESULTS (VALUES ARE IN TERMS OF MILLISECONDS)

3) Convergence of the Multiplicative Auction Algorithm: In this part, we address the question of how quickly the `multiplicative auction` algorithm converges to the exact solution. For this purpose, we executed the algorithm for different values of ϵ from 0.1 to 0.95 and calculated the average approximate ratio over all tests. The result is shown in Figure IV-C3.

The first observation is that the average approximation ratio is consistently above 0.9. This demonstrates that the `multiplicative auction` algorithm performs very well in practice, even for larger values of ϵ. Another observation is that the approximation ratio converges to 1 as ϵ approaches 0. This is theoretically logical because it is established that the algorithm is $1 - \epsilon$ approximate. However, as observed, the approximation ratio does not consistently decrease with

decreasing ϵ. This is because $1-\epsilon$ serves as a lower bound for the algorithm's approximation ratio, and decreasing ϵ does not guarantee improved performance across all tests. Therefore, this observation does not contradict the proven theorems about the algorithm.

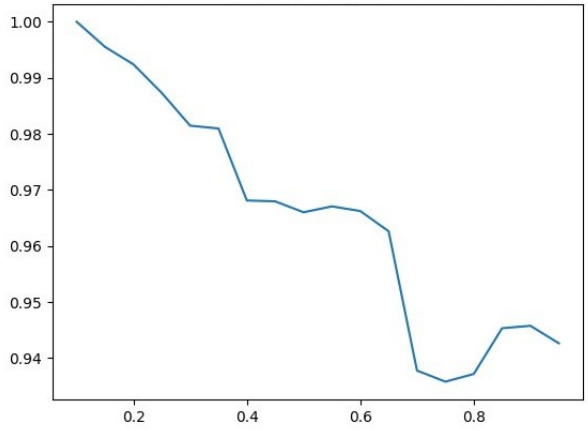

Fig. 2. Average approximation ratio for different ϵ values

V. Conclusion

In this report, we analyzed the performance of the very recent LP-based algorithm for bipartite maximum matching, the multiplicative auction algorithm. It can be seen that this algorithm executes and terminates very quickly compared to previous exact and approximate algorithms. Furthermore, it not only produces $1-\epsilon$ approximate solutions but also performs admirably in practice, even for relatively large values of ϵ. Additionally, despite being based on the matching linear program conceptually, its implementation does not directly involve LP solving and is relatively straightforward.

This suggests that optimization methods can play a crucial role in addressing fundamental computer science problems and can lead to the development of improved algorithms. However, several open questions remain, paving the way for further research:

- Our results demonstrate that the algorithm generally performs well for large ϵ values in practice. However, this raises the question of whether there exists a general class of graphs for which the algorithm consistently produces $(1-\epsilon)$-approximate solutions, or if further analysis can yield better lower bounds for the algorithm's performance.
- Can the underlying idea of the multiplicative auction algorithm be extended to address the maximum matching problem in general graphs? While the latest result for general graphs is attributed to Duan and Pettie [4], its implementation is notably complex and includes an additional $log(\epsilon^{-1})$ factor in its time complexity.
- Is it possible to improve the time complexity of the multiplicative auction algorithm to develop a more efficient algorithm?

References

1 Chekuri, C. and Quanrud, K., "Randomized MWU for positive lps," in *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, Czumaj, A., Ed. SIAM, 2018, pp. 358–377. [Online]. Available: https://doi.org/10.1137/1.9781611975031.25

2 Allen-Zhu, Z. and Orecchia, L., "Nearly linear-time packing and covering LP solvers - achieving width-independence and -convergence," *Math. Program.*, vol. 175, no. 1-2, pp. 307–353, 2019. [Online]. Available: https://doi.org/10.1007/s10107-018-1244-x

3 Zheng, D. W. and Henzinger, M., "Multiplicative auction algorithm for approximate maximum weight bipartite matching," in *Integer Programming and Combinatorial Optimization - 24th International Conference, IPCO 2023, Madison, WI, USA, June 21-23, 2023, Proceedings*, ser. Lecture Notes in Computer Science, Pia, A. D. and Kaibel, V., Eds., vol. 13904. Springer, 2023, pp. 453–465. [Online]. Available: https://doi.org/10.1007/978-3-031-32726-1_32

4 Duan, R. and Pettie, S., "Linear-time approximation for maximum weight matching," *J. ACM*, vol. 61, no. 1, pp. 1:1–1:23, 2014. [Online]. Available: https://doi.org/10.1145/2529989

5 Kuhn, H. W., "The hungarian method for the assignment problem," in *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, Jünger, M., Liebling, T. M., Naddef, D., Nemhauser, G. L., Pulleyblank, W. R., Reinelt, G., Rinaldi, G., and Wolsey, L. A., Eds. Springer, 2010, pp. 29–47. [Online]. Available: https://doi.org/10.1007/978-3-540-68279-0_2

6 Chen, L., Kyng, R., Liu, Y. P., Peng, R., Gutenberg, M. P., and Sachdeva, S., "Maximum flow and minimum-cost flow in almost-linear time," in *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*. IEEE, 2022, pp. 612–623. [Online]. Available: https://doi.org/10.1109/FOCS54457.2022.00064

7 Koufogiannakis, C. and Young, N. E., "A nearly linear-time PTAS for explicit fractional packing and covering linear programs," *Algorithmica*, vol. 70, no. 4, pp. 648–674, 2014. [Online]. Available: https://doi.org/10.1007/s00453-013-9771-6

8 Kapralov, M., "Space lower bounds for approximating maximum matching in the edge arrival model," in *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, Marx, D., Ed. SIAM, 2021, pp. 1874–1893. [Online]. Available: https://doi.org/10.1137/1.9781611976465.112

9 Assadi, S., Liu, S. C., and Tarjan, R. E., "An auction algorithm for bipartite matching in streaming and massively parallel computation models," in *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, Le, H. V. and King, V., Eds. SIAM, 2021, pp. 165–171. [Online]. Available: https://doi.org/10.1137/1.9781611976496.18

10 Schrijver, A. *et al.*, *Combinatorial optimization: polyhedra and efficiency*. Springer, 2003, vol. 24.

11 Drake, D. E. and Hougardy, S., "A simple approximation algorithm for the weighted matching problem," *Inf. Process. Lett.*, vol. 85, no. 4, pp. 211–213, 2003. [Online]. Available: https://doi.org/10.1016/S0020-0190(02)00393-9

12 Bertsekas, D. P., "A new algorithm for the assignment problem," *Math. Program.*, vol. 21, no. 1, pp. 152–171, 1981. [Online]. Available: https://doi.org/10.1007/BF01584237

13 Demange, G., David, G., and Sotomayor, M., "Multi-item auctions," HAL, Post-Print, 1986. [Online]. Available: https://EconPapers.repec.org/RePEc:hal:journl:halshs-00670982