

IMPROVED APPROXIMATION GUARANTEES FOR WEIGHTED MATCHING IN THE SEMI-STREAMING MODEL

LEAH EPSTEIN¹ AND ASAF LEVIN² AND JULIÁN MESTRE³ AND DANNY SEGEV⁴

¹ Department of Mathematics, University of Haifa, 31905 Haifa, Israel.

² Chaya fellow. Faculty of Industrial Engineering and Management, The Technion, 32000 Haifa, Israel.

³ Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany.

⁴ Department of Statistics, University of Haifa, 31905 Haifa, Israel.

ABSTRACT. We study the maximum weight matching problem in the semi-streaming model, and improve on the currently best one-pass algorithm due to Zelke (Proc. STACS '08, pages 669–680) by devising a deterministic approach whose performance guarantee is $4.91 + \varepsilon$. In addition, we study *preemptive* online algorithms, a sub-class of one-pass algorithms where we are only allowed to maintain a feasible matching in memory at any point in time. All known results prior to Zelke's belong to this sub-class. We provide a lower bound of 4.967 on the competitive ratio of any such deterministic algorithm, and hence show that future improvements will have to store in memory a set of edges which is not necessarily a feasible matching. We conclude by presenting an empirical study, conducted in order to compare the practical performance of our approach to that of previously suggested algorithms.

1. Introduction

The computational task of detecting maximum weight matchings is one of the most fundamental problems in discrete optimization, attracting plenty of attention from the operations research, computer science, and mathematics communities. (For a wealth of references on matching problems see [16].) In such settings, we are given an undirected graph $G = (V, E)$ whose edges are associated with non-negative weights specified by $w : E \rightarrow \mathbb{R}_+$. A set of edges $M \subseteq E$ is a *matching* if no two of the edges share a common vertex, that is, the degree of any vertex in (V, M) is at most 1. The weight $w(M)$ of a matching M is defined as the combined weight of its edges, i.e., $\sum_{e \in M} w(e)$. The objective is to compute a matching of maximum weight. We study this problem in two related computational models: the *semi-streaming* model and the *preemptive online* model.

The semi-streaming model. Even though these settings appear to be rather simple as first glance, it is worth noting that matching problems have an abundance of flavors, usually depending on how the input is specified. In this paper, we investigate weighted matchings in the *semi-streaming* model, first suggested by Muthukrishnan [14]. Specifically, a *graph stream* is a sequence e_{i_1}, e_{i_2}, \dots of

The third author was supported by an Alexander von Humboldt Fellowship.



distinct edges, where e_{i_1}, e_{i_2}, \dots is an arbitrary permutation of E . When an algorithm is processing the stream, edges are revealed sequentially, one at a time. Letting $n = |V|$ and $m = |E|$, efficiency in this model is measured by the space $S(n, m)$ a graph algorithm uses, the time $T(n, m)$ it requires to process each edge, and the number of passes $P(n, m)$ it makes over the input stream. Throughout the paper, however, we focus on one-pass algorithms, that is, $P(n, m) = 1$. The main restriction is that the space $S(n, m)$ is limited to $O(n \cdot \text{polylog}(n))$ bits of memory. We refer the reader to a number of recent papers [14, 4, 5, 2, 12] and to the references therein for a detailed literature review.

Online version. Online matching has previously been modeled as follows [9]. Edges are presented one by one to the algorithm, along with their weight. Once an edge is presented, we must make an irrevocable decision, whether to accept it or not. An edge may be accepted only if its addition to the set of previously accepted edges forms a feasible matching. In other words, an algorithm must keep a matching at all times, and its final output consists of all edges which were ever accepted. In this model, it is easy to verify that the competitive ratio of any (deterministic or randomized) algorithm exceeds any function of the number of vertices, meaning that no competitive algorithm exists. However, if all weights are equal, a greedy approach which accepts an edge whenever possible, has a competitive ratio of 2, which is best possible for deterministic algorithms [9].

Similarly to other online settings (such as call control problems [6]), a preemptive model can be defined, allowing us to remove a previously accepted edge from the current matching at any point in time; this event is called *preemption*. Nevertheless, an edge which was either rejected or preempted cannot be inserted to the matching later on. We point out that other types of online matching problems were studied as well [9, 7, 10, 1].

Comparison between the models. Both one-pass semi-streaming algorithms and online algorithms perform a single scan of the input. However, unlike semi-streaming algorithms, online algorithms are allowed to concurrently utilize memory for two different purposes. The first purpose is obviously to maintain the current solution, which must always be a feasible matching, implying that the memory size of this nature is bounded by the maximal size of a matching. The second purpose is to keep track of arbitrary information regarding the past, without any concrete bound on the size of memory used. Therefore, in theory, online algorithms are allowed to use much larger memory than is allowed in the semi-streaming model. Moreover, although this possibility is rarely used, online algorithms may perform exponential time computations whenever a new piece of input is revealed. On the other hand, a semi-streaming algorithm may re-insert an edge to the current solution, even if it has been temporarily removed, as long as this edge was kept in memory. This extra power is not allowed for online (preemptive) algorithms, making them inferior in this sense in comparison to their semi-streaming counterparts.

Previous work. Feigenbaum et al. [4] were the first to study matching problems under similar assumptions. Their main results in this context were a semi-streaming algorithm that computes a $(3/2 + \varepsilon)$ -approximation in $O(\log(1/\varepsilon)/\varepsilon)$ passes for maximum cardinality matching in bipartite graphs, as well as a one-pass 6-approximation for maximum weighted matching in arbitrary graphs. Later on, McGregor [12] improved on these findings, to obtain performance guarantees of $(1 + \varepsilon)$ and $(2 + \varepsilon)$ for the maximum cardinality and maximum weight versions, respectively, being able to handle arbitrary graphs with only a constant number of passes (depending on $1/\varepsilon$). In addition, McGregor [12] tweaked the one-pass algorithm of Feigenbaum et al. into achieving a ratio of 5.828. Finally, Zelke [17] has recently attained an improved approximation factor of 5.585, which stands as the currently best one-pass algorithm. Note that the 6-approximation algorithm in [4] and the 5.828-approximation algorithm in [12] are preemptive online algorithms. On the other hand, the algorithm

of Zelke [17] uses the notion of shadow-edges, which may be re-inserted into the matching, and hence it is not an online algorithm.

Main result I. The first contribution of this paper is to improve on the above-mentioned results, by devising a deterministic one-pass algorithm in the semi-streaming model, whose performance guarantee is $4.91 + \varepsilon$. In a nutshell, our approach is based on partitioning the edge set into $O(\log n)$ weight classes, and computing a separate maximal matching for each such class in online fashion, using $O(n \cdot \text{polylog}(n))$ memory bits overall. The crux lies in proving that the union of these matchings contains a single matching whose weight compares favorably to the optimal one. The specifics of this algorithm are presented in Section 2.

Main result II. Our second contribution is motivated by the relation between semi-streaming algorithms and *preemptive* online algorithms, which must maintain a feasible matching at any point in time. To our knowledge, there are currently no lower bounds on the competitive ratio that can be achieved by incorporating preemption. Thus, we also provide a lower bound of 4.967 on the performance guarantee of any such deterministic algorithm. As a result, we show that improved one pass algorithms for this problem must store more than just a matching in memory. Further details are provided in Section 3.

Main result III. We conclude with the first ever experimental study in the context of semi-streaming algorithms for matching problems, conducted in order to compare the practical performance of our approach to that of previously suggested algorithms. In Section 4, we demonstrate that by carefully calibrating some cut-off parameters, combined with the idea of running multiple algorithms in parallel, one can achieve practical performance guarantees that far exceed theoretical ones, at least when real-life instances are considered.

2. The Semi-Streaming Algorithm

This section is devoted to obtaining main result I, that is, an improved one-pass algorithm for the weighted matching problem in the semi-streaming model. We begin by presenting a simple deterministic algorithm with a performance guarantee of 8. We then show how to randomize its parameters, still within the semi-streaming framework, and obtain an expected approximation ratio of 4.9108. Finally, we de-randomize the algorithm by showing how to emulate the required randomness using multiple copies (constant number) of the deterministic algorithm, while paying an additional additive factor of at most ε , for any fixed $\varepsilon > 0$.

2.1. A simple deterministic approach

Preliminaries. We maintain the maximum weight of any edge w_{\max} seen so far in the input stream. Clearly, the maximum weight matching of the edges seen so far has weight in the interval $[w_{\max}, \frac{n}{2}w_{\max}]$. We denote a maximum weight matching and its cost by OPT ; it should be clear which one is meant from the context. Note that if we disregard all edges with weight at most $\frac{2\tilde{\varepsilon}w_{\max}}{n}$, the weight of the maximum weight matching in the resulting instance decreases by an additive term of at most $\tilde{\varepsilon}w_{\max} \leq \tilde{\varepsilon}\text{OPT}$.

Our algorithm has a parameter $\gamma > 1$, and a value $\phi > 0$. We define weight classes of edges in the following way. For every $i \in \mathbb{Z}$, we let the class W_i be the collection of edges whose weight is in the interval $[\phi\gamma^i, \phi\gamma^{i+1})$. We note that by our initial assumption, the weight of each edge is in the interval $[\frac{2\tilde{\varepsilon}w_{\max}}{n}, w_{\max}]$, and we say that a weight class W_i is *under consideration* if its weight

interval $[\phi\gamma^i, \phi\gamma^{i+1})$ intersects $[\frac{2\epsilon w_{\max}}{n}, w_{\max}]$. The number of classes which are under consideration at any point in time is $O(\log_\gamma \frac{n}{\epsilon})$.

The algorithm. Our algorithm simply maintains the list of classes under consideration and maintains a maximal (unweighted) matching for each such class. In other words, when the value of w_{\max} changes, we delete from the memory some of these matchings, corresponding to classes that are no longer under consideration. Note that to maintain a maximal matching in a given subgraph, we only need to check if the two endpoints of the new edge are not covered by existing edges of the matching.

To conclude, for every new edge $e \in E$ we proceed as follows. We first check if $w(e)$ is greater than the current value of w_{\max} . If so, we update w_{\max} and the list of weight classes under consideration accordingly. Then, we find the weight class of $w(e)$, and try to extend its corresponding matching; i.e., e will be added to this matching if it remains a matching after doing so.

Note that at each point the content of the memory is comprised of a fixed number of parameter values and a collection of $O(\log_\gamma \frac{n}{\epsilon})$ matchings, consisting of $O(n \log_\gamma \frac{n}{\epsilon})$ edges overall. Therefore, our algorithm indeed falls in the semi-streaming model.

At the conclusion of the input sequence, we need to return a single matching rather than a collection of matchings. To this end, we could compute a maximum weighted matching of the edges in the current memory. However, for the specific purposes of our analysis, we use the following faster algorithm. We sort the edges in memory in decreasing order of weight classes, such that the edges in W_i appear before those in W_{i-1} , for every i . Using this sorted list of edges, we apply a greedy algorithm for selecting a maximal matching, in which the current edge is added to this matching if it remains a matching after doing so. Then, the post-processing time needed is linear in the size of the memory used, that is, $O(n \log_\gamma \frac{n}{\epsilon})$. This concludes the presentation of the algorithm and its implementation as a semi-streaming algorithm.

Analysis. For purposes of analysis, we round down the weight of each edge so that the weight of all edges in W_i equals $\phi\gamma^i$. This way, we obtain *rounded* edge weights. For our optimal solution OPT let us denote by OPT' its rounded weight. The next claim follows from the definition of W_i .

Lemma 2.1. $\text{OPT} \leq \gamma \text{OPT}'$.

As an intermediate step, we analyze an improved algorithm that keeps all weight classes. That is, for each i , we use M_i to denote the maximal matching of class W_i at the end of the input, and denote by M the solution obtained by this algorithm, if we would have applied it. Similarly, we denote by OPT_i the set of edges in OPT which belong to W_i . For every i , we define the set of vertices P_i , associated with W_i , to be the set of endpoints of edges in M_i that are not associated with higher weight classes:

$$P_i = \{u, v \mid (u, v) \in M_i\} \setminus (P_{i+1} \cup P_{i+2} \cup \dots).$$

For a vertex $p \in P_i$, we define its associated weight to be $\phi\gamma^i$. For vertices which do not belong to any P_i , we let their associated weight be zero. We next bound the total associated weight of all vertices.

Lemma 2.2. *The total associated weight of all vertices is at most $\frac{2\gamma}{\gamma-1} \cdot w(M)$.*

Proof. Consider a vertex $u \in P_i$ and let (u, v) be the edge in M_i adjacent to u . If $(u, v) \in M$ then we charge the weight associated with u to the edge (u, v) . Thus, an edge $e \in M_i$ is charged at most twice from vertices associated with its own weight class. Otherwise, if $(u, v) \notin M$ then there must be some other edge $e \in M \cap M_j$, for some $j > i$, that prevented us from adding (u, v) to M , in which case we charge the weight associated with u to e . Notice that $u \notin e$, or otherwise, u would not be

associated with W_i . Thus, the edge $e \in M_j$ must be of the form $e = (v, x)$ and can only be charged twice from vertices in weight class i , once through v and once through x .

To bound the ratio between $w(M)$ and the total associated weight of the vertices, it suffices to bound the ratio between the weight of an edge $e \in M$ and the total associated weight of the vertices which are charged to e . Assume that $e \in M_j$, then there are at most two vertices which are charged to e and class i for all $i \leq j$, and no vertex is associated to e and class i for $i > j$. Hence, the total associated weight of these vertices is at most

$$2 \sum_{i \leq j} \phi \gamma^i < 2\phi \gamma^j \cdot \sum_{i=0}^{\infty} \frac{1}{\gamma^i} = 2\phi \gamma^j \cdot \frac{1}{1 - 1/\gamma} = \phi \gamma^j \cdot \frac{2\gamma}{\gamma - 1},$$

and the claim follows since $w(e) \geq \phi \gamma^j$. ■

It remains to bound opt' with respect to the total associated weight.

Lemma 2.3. *The total weight associated with all vertices is at most opt' .*

Proof. It suffices to show that for every edge $e = (x, y) \in \text{opt}'_i$ the maximum of the associated weights of x and y is at least the rounded weight of e . Suppose that this claim does not hold, then x and y are not covered by M_i , as otherwise their associated weight would be at least $\phi \gamma^i$. Hence, when the algorithm considered e , we would have added e to M_i , contradicting our assumption that x and y are not covered by M_i . ■

Now instead of considering all weight categories, we construct the matching M only using edges with weight at least $\frac{2\tilde{\varepsilon}w_{\max}}{n}$. Using the above sequence of lemmas, and recalling that we lose another $\frac{1}{1-\tilde{\varepsilon}}$ factor in the approximation ratio due to disregarding these cheap edges, we obtain the following inequality:

$$\text{OPT} \leq \gamma \text{OPT}' \leq \frac{1}{1 - \tilde{\varepsilon}} \cdot \frac{2\gamma^2}{\gamma - 1} \cdot w(M). \quad (2.1)$$

For any $\varepsilon > 0$, setting $\tilde{\varepsilon} = \frac{\varepsilon}{(2\gamma^2/(\gamma-1)) + \varepsilon}$, we get an approximation ratio of $\frac{2\gamma^2}{\gamma-1} + \varepsilon$. This ratio is optimized for $\gamma = 2$, where it equals $(8 + \varepsilon)$. Hence, we have established the following theorem.

Theorem 2.4. *For any fixed $\varepsilon > 0$, there is a deterministic one-pass semi-streaming algorithm whose approximation ratio is $8 + \varepsilon$.*

2.2. Improved approximation ratio through randomization

In what follows, we analyze a randomized variant of the deterministic algorithm which was presented in the previous subsection. In general, this variant sets the value of ϕ to be $\phi = \gamma^\delta$ where δ is a random variable. This method is commonly referred to as *randomized geometric grouping* [8].

Formally, let δ be a continuous random variable which is uniformly distributed on the interval $[0, 1)$. We define the weight class $W_i(\delta)$ to be the edges whose weight is in the interval $[\gamma^{i+\delta}, \gamma^{i+1+\delta})$, and run the algorithm as in the previous subsection. Note that this algorithm uses only the partition of the edges into classes and not the precise values of their weights. In addition, we denote by $M(\delta)$ the resulting matching obtained by the algorithm, and by $TW(\delta)$ the total associated weight of the vertices, where for a vertex $p \in P_i$ we define its associated weight to be $\gamma^{i+\delta}$; i.e., the minimal value in the interval defining $W_i(\delta)$. We also denote by $\text{opt}'(\delta)$ the value of opt' for this particular δ .

For any fixed value of δ , inequality (2.1) immediately implies $\text{opt}'(\delta) \leq \left(\frac{2\gamma}{\gamma-1} + \varepsilon\right) \cdot w(M(\delta))$. Note that $\text{opt}'(\delta)$ and $w(M(\delta))$ are random variables, such that for each realization of δ the above

inequality holds. Hence, this inequality holds also for their expected values. That is, we have established the following lemma where $E_\delta[\cdot]$ represents expectation with respect to the random variable δ .

Lemma 2.5. $E_\delta[\text{OPT}'(\delta)] \leq \left(\frac{2\gamma}{\gamma-1} + \varepsilon\right) \cdot E_\delta[w(M(\delta))]$.

We next upper bound OPT in terms of $E_\delta[\text{OPT}'(\delta)]$.

Lemma 2.6. $\frac{\gamma \ln \gamma}{\gamma-1} \cdot E_\delta[\text{OPT}'(\delta)] \geq \text{OPT}$.

Proof. We will show the corresponding inequality for each edge $e \in \text{OPT}$. We denote by $w'_\delta(e)$ the rounded weight of e for a specific value of δ . Then, it suffices to show that $\frac{\gamma \ln \gamma}{\gamma-1} \cdot E_\delta[w'_\delta(e)] \geq w(e)$. Let p be an integer, and let $0 \leq \alpha < 1$ be the value that satisfies $w(e) = \gamma^{p+\alpha}$. Then, for $\delta \leq \alpha$, $w'_\delta(e) = \gamma^{p+\delta}$, and for $\delta > \alpha$, $w'_\delta(e) = \gamma^{p-1+\delta}$, thus the expected rounded weight of e over the choices of δ is

$$E_\delta[w'_\delta(e)] = \int_0^\alpha \gamma^{p+\delta} d\delta + \int_\alpha^1 \gamma^{p-1+\delta} d\delta = \frac{1}{\ln \gamma} \cdot (\gamma^p(\gamma^\alpha - 1) + \gamma^{p-1}(\gamma - \gamma^\alpha)) = w(e) \cdot \left(1 - \frac{1}{\gamma}\right) \frac{1}{\ln \gamma},$$

and the claim follows. \blacksquare

Combining the above two lemmas we obtain that the expected weight of the resulting solution is at least $\left(\frac{(\gamma-1)^2}{2\gamma^2 \ln \gamma} + \varepsilon\right) \cdot \text{OPT}$. This approximation ratio is optimized for $\gamma \approx 3.513$, where it is roughly $(4.9108 + \varepsilon)$. Hence, we have established the following theorem.

Theorem 2.7. *For any fixed $\varepsilon > 0$, there is a randomized one-pass semi-streaming algorithm whose expected approximation ratio is roughly $4.9108 + \varepsilon$.*

2.3. Derandomization

Prior to presenting our de-randomization, we slightly modify the randomized algorithm of the previous subsection. In this variation, instead of picking δ uniformly at random from the interval $[0, 1)$ we pick δ' uniformly at random from the discrete set $\left\{0, \frac{1}{q}, \frac{2}{q}, \dots, \frac{q-1}{q}\right\}$, where q is a parameter whose value will be determined later. We apply the same method as in the previous section, replacing δ by δ' . Then, using Lemma 2.5, we obtain $E_{\delta'}[\text{OPT}'(\delta')] \leq \left(\frac{2\gamma}{\gamma-1} + \varepsilon\right) \cdot E_{\delta'}[w(M(\delta'))]$. To extend Lemma 2.6 to this new setting, we note that δ' can be obtained by first picking δ and then rounding it down to the largest number in $\left\{0, \frac{1}{q}, \frac{2}{q}, \dots, \frac{q-1}{q}\right\}$ which is at most δ . In this way, we couple the distributions of δ and δ' . Now consider the rounded weight of an edge e in OPT in the two distinct values of δ and δ' . The ratio between the two rounded weights is at most $\gamma^{1/q}$. Therefore, we establish that $\frac{\gamma \ln \gamma}{\gamma-1} \cdot \gamma^{1/q} \cdot E_{\delta'}[\text{OPT}'(\delta')] \geq \text{OPT}$, and the resulting approximation ratio of the new variation is $\frac{2\gamma^{2+1/q} \ln \gamma}{(\gamma-1)^2} + \varepsilon$. By setting q to be large enough (picking $q = \lceil \log_\gamma^{-1}(\varepsilon/5) \rceil$ is sufficient), the resulting approximation ratio is bounded by $\frac{2\gamma^2 \ln \gamma}{(\gamma-1)^2} + 2\varepsilon$.

De-randomizing the new variation in the semi-streaming model is straightforward. We simply run in parallel all q possible outcomes of the algorithm, one for each possible value of δ' , and pick the best solution among the q solutions obtained. Since q is a constant (for fixed values of ε), the resulting algorithm is still a semi-streaming algorithm whose performance guarantee is $4.9108 + 2\varepsilon$. By scaling ε prior to applying the algorithm, we establish the following result.

Theorem 2.8. *For any fixed $\varepsilon > 0$, there is a deterministic one-pass semi-streaming $(4.9108 + \varepsilon)$ -approximation algorithm for the weighted matching problem. This algorithm processes each input edge in constant time and required $O(n)$ time at the end of the input to compute the final output.*

3. Online Preemptive Matching

In this section, we establish the following theorem.

Theorem 3.1. *The competitive ratio of any deterministic preemptive online algorithm is at least $\mathcal{R} - \varepsilon$ for any $\varepsilon > 0$, where $\mathcal{R} \approx 4.967$ is the unique real solution of the equation $x^3 = 4(x^2 + x + 1)$.*

Recall that the algorithm of Feigenbaum et al. [4] and that of McGregor [12] can be viewed as online preemptive algorithms; their competitive ratios are 6 and 5.828, respectively.

Definitions and properties. Let $C = \mathcal{R} - \varepsilon$ for some arbitrary but fixed $\varepsilon > 0$. Our goal is to show that the competitive ratio of any deterministic algorithm is at least C . To this end, we construct an input graph iteratively. In the construction of the input, edge weights come from two sequences. The main sequence w_1, w_2, \dots , and the additional sequence is w'_1, w'_2, \dots , are defined as follows:

$$w_i = \begin{cases} 1 & i=1 \\ \frac{1}{2C+1} \left((C^2 + 1)w_{i-1} - C \sum_{j=1}^{i-2} w_j \right) & i>1 \end{cases} \quad w'_i = \begin{cases} 1 & i=1 \\ \frac{1}{C} \left((C + 1)w_i - w_{i-1} \right) & i>1 \end{cases} \quad (3.1)$$

The sequences are defined according to (3.1) as long as $w_{i-1} \geq w_{i-2}$. As soon as $w_{n-1} < w_{n-2}$ for some n , both sequences stop with w_n and w'_n , respectively. In the full version [3] of this paper we show that the sequences are well defined in the sense that they indeed have finite length. Let $S_i = \sum_{j=1}^i w_j$ and $S_0 = 0$.

From the definition (3.1) and simple algebra, one can derive the following properties of these sequences. We omit their justification due to lack of space.

Property 1. For all $i = 1, \dots, n - 2$ we have $w_i \leq w'_i$, but $w_{n-1} > w'_{n-1}$.

Property 2. For all $i = 1, \dots, n - 2$ we have $Cw_i = S_{i-1} + w_{i+1} + w'_{i+1}$.

Property 3. For all $i = 1, \dots, n - 2$ we have $Cw'_i = S_{i-2} + w_i + w_{i+1} + w'_{i+1}$.

Input construction, step 1. To better understand our construction, we advise the reader to consult Figure 1. The input is created in n steps. In the initial step, two edges (a_1, x_1) and (b_1, x_1) , each of weight w_1 , are introduced. Assume that after both edges have arrived, the online algorithm keeps the edge (a_1, x_1) .

Every future step can be of two distinct types, which will be described later on. We maintain the following invariants throughout the construction.

Invariant 1. Immediately after the i th step, the set $M_i = \{(x_1, b_1), \dots, (x_i, b_i)\}$ forms a matching.

Invariant 2. Immediately after the i th step, the algorithm keeps a single edge e_i , which can be one of two edges:

- i) If $e_i = (x_i, a_i)$ then its weight is w_i and a_i is unmatched in M_i .
- ii) If $e_i = (y_i, c_i)$ then its weight is w'_i , $y_i = x_j$ for some $j < i$, and c_i is unmatched in M_i .

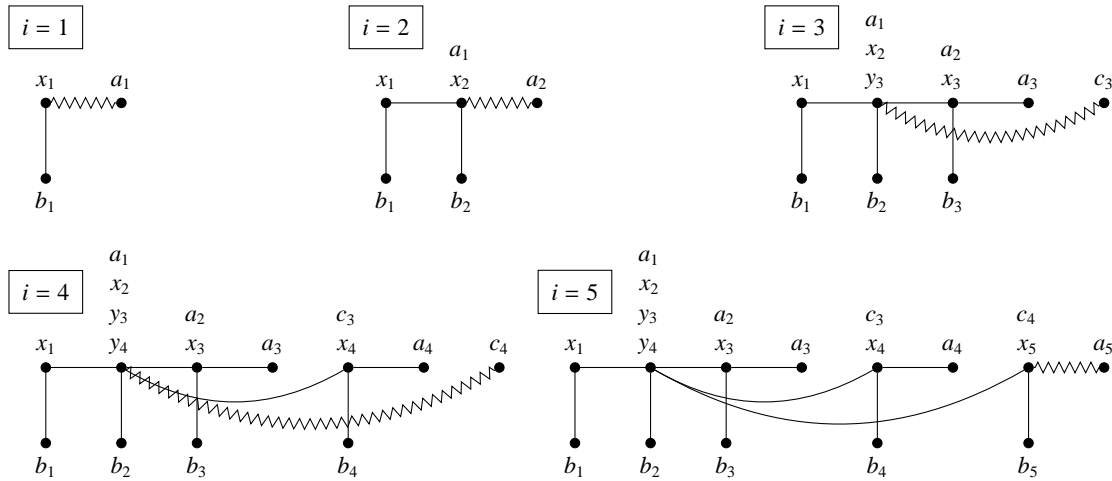


Figure 1: An example of five steps of the lower bound construction. The curved edges denote the edge kept by the online algorithm at each time. In the first two steps, the edges (x_i, a_i) are chosen by the algorithm. In the third step, (x_3, a_3) is not chosen by the algorithm, so (y_3, c_3) arrives next. In the fourth step, (x_4, a_4) is not chosen by the algorithm, so (y_4, c_4) arrives next. In the fifth step, (x_5, a_5) is chosen by the algorithm, so no further edges arrive in this step.

The invariants clearly holds after the first step: The algorithm keeps (x_1, a_1) and a_1 is free in $M_1 = \{(x_1, b_1)\}$. We next define the subsequent steps and show that the invariants holds throughout.

Input construction, step $i + 1 < n$. We now show how to construct the edges of step $i + 1$, for the case $i + 1 < n$. We introduce two new edges of weight w_{i+1} . Let x_{i+1} be a_i if $e_i = (x_i, a_i)$, and x_{i+1} be c_i if $e_i = (y_i, c_i)$. The new edges are (x_{i+1}, b_{i+1}) , and (x_{i+1}, a_{i+1}) , where a_{i+1} and b_{i+1} are new vertices. According to Invariant 2, the vertex x_{i+1} is unmatched in M_i . It follows that M_{i+1} is a matching and thus Invariant 1 holds in this step. Both edges have a common endpoint with the edge that the algorithm has, and the algorithm can either preempt e_i , in which case we assume (without loss of generality) that it now has (x_{i+1}, a_{i+1}) , or else it keeps the previous edge. If the algorithm holds onto e_i then let y_{i+1} be x_i if $e_i = (x_i, a_i)$, and y_{i+1} be y_i if $e_i = (y_i, c_i)$. In this case a third edge, (y_{i+1}, c_{i+1}) , with weight of w'_{i+1} is introduced. The vertex c_{i+1} is new. There are four cases to consider depending on which edge the algorithm had at the end of the i th step and whether it is preempted right away or not.

In the first case, the algorithm has $e_i = (x_i, a_i)$ at the end of the i th step and replaces it with $(x_{i+1}, a_{i+1}) = (a_i, a_{i+1})$. Since a_{i+1} is a new vertex (and different than x_{i+1}) it follows that a_{i+1} is free in M_{i+1} . Thus, case i) of Invariant 2 holds.

In the second case, the algorithm has $e_i = (y_i, c_i)$ at the end of the i th step and it replaces it with $(x_{i+1}, a_{i+1}) = (c_i, a_{i+1})$. It follows that a_{i+1} is free in M_{i+1} . Thus, case ii) of Invariant 2 holds.

For the remaining two cases note that if $w'_i \leq 0$ or $w_i < 0$ and the algorithm has a single edge of weight w'_i or w_i , respectively, then the optimal solution is strictly positive and the value of the algorithm is non-positive, hence the resulting approximation ratio in this case is unbounded.

Consequently, we can assume without loss of generality that if the algorithm has a single edge at the end of step i then its weight is strictly positive.

Now consider the case where the algorithm has $e_i = (a_i, x_i)$ at the end of the i th step but does not replace it with $(x_{i+1}, a_{i+1}) = (a_i, a_{i+1})$. If this happens, we show that the algorithm must replace the edge with $(y_{i+1}, c_{i+1}) = (x_i, c_{i+1})$. Assume that this is not the case. Then the profit of the algorithm is w_i . Consider the solution $M_{i+1} - (x_i, b_i) + (y_{i+1}, c_{i+1})$. The cost of this matching is $S_{i+1} - w_i + w'_{i+1}$, which equals Cw_i , by Property 2. In other words, the solution kept by the algorithm is C -competitive. Since our goal is to prove precisely this, we can assume this event never happens. Thus, the algorithm must switch to the edge (y_{i+1}, c_{i+1}) , which leads us to case ii) of Invariant 2.

Finally, consider the case where the algorithm has $e_i = (y_i, c_i)$ at the end of the i th step but does not replace it with $(x_{i+1}, a_{i+1}) = (c_i, a_{i+1})$. If this happens, we show that the algorithm must replace the edge with $(y_{i+1}, c_{i+1}) = (y_i, c_{i+1})$. Assume that this is not the case. Then the profit of the algorithm is w'_i . Consider the solution $M_{i+1} - (x_j, b_j) + (y_{i+1}, c_{i+1})$, where $j < i$ is the index from case ii) in Invariant 2 that corresponds to e_i . The cost of this matching is at least $S_{i+1} - w_{i-1} + w'_{i+1}$, which equals Cw'_i , by Property 3. As in the previous case, we can assume this never happens. Thus, the algorithm must switch to the edge (x_{i+1}, a_{i+1}) , which leads us to case ii) of Invariant 2.

This finishes the description of the input graph construction, as well as the justification that Invariants 1 and 2 hold at each step along the way.

Bounding the competitive ratio. We next define a recursive formula for S_i . By definition (3.1) of the sequence w_i , we have

$$\begin{cases} S_0 = 0 \\ S_1 = 1 \\ S_{k+1} = \frac{C^2+2C+2}{2C+1}S_k - \frac{C^2+C+1}{2C+1}S_{k-1}, \quad \text{for } k \geq 1 \end{cases} \quad (3.2)$$

Lemma 3.2. *There exists a value of n such that $w_{n-2} > w_{n-1}$; for this value, $\frac{S_{n-1}}{w_{n-1}} > C$ holds.*

Proof. The first claim is proved by solving the recurrence (3.2), using standard tools [3]. To prove the second part, note that $w_{n-2} > w_{n-1}$ is equivalent to $S_{n-1} - S_{n-2} < S_{n-2} - S_{n-3}$. Hence using the recursive formula we conclude that

$$S_{n-1} - 2S_{n-2} + \frac{2C+1}{-C^2-C-1}S_{n-1} + \frac{C^2+2C+2}{C^2+C+1}S_{n-2} < 0,$$

that is,

$$S_{n-1} \cdot (C^2 + C + 1 - 2C - 1) + S_{n-2} \cdot (C^2 + 2C + 2 - 2C^2 - 2C - 2) < 0,$$

which is equivalent to $(C^2 - C)S_{n-1} - C^2S_{n-2} < 0$, so $C(S_{n-1} - S_{n-2}) < S_{n-1}$, and we conclude that $Cw_{n-1} < S_{n-1}$, as claimed. \blacksquare

Everything is in place to prove the main claim of this section.

Proof of Theorem 3.1. From Invariant 2, we conclude that at the end of iteration $n-1$, the algorithm only has the edge e_{n-1} , which can have weight w'_{n-1} or w_{n-1} . From Property 1, it follows that $\max\{w_{n-1}, w'_{n-1}\} = w_{n-1}$. On the other hand, from Invariant 1, we know that there is a matching with cost S_{n-1} . Therefore, the competitive ratio of any algorithm is at least $\frac{S_{n-1}}{w_{n-1}}$. From Lemma 3.2 we then conclude that the competitive ratio is at least C . \blacksquare

4. Experimental Evaluation

In this section, we present the results of an empirical study, conducted in order to compare the practical performance of our approach to that of previously suggested algorithms. More specifically, the complete set of algorithms that have been implemented and extensively tested can be briefly listed as follows:

- **LAYERED**: The algorithm described in Section 2, which keeps $O\left(\log \frac{n}{\epsilon}\right)$ matchings.
- **ONLINE**: The algorithm of McGregor [12], based on keeping a single matching at all times.
- **SHADOW**: The algorithm of Zelke [17], with two shadow edges for each matching edge.

For **LAYERED** and **SHADOW**, we made use of an addition optimization phase, in which a maximum weight matching is computed among the edges that were kept in memory. The main reason for this extra effort is that we were interested in determining the best possible practical performance that can be extracted out of these algorithms, rather than in worst case performance and nothing more. We point out that this phase is performed only once, and that one can always employ a linear-time approximation [11] should running time be a concern.

Special features. Each of above-mentioned algorithms is parameterized. Typically, this parameter is chosen to minimize the worst-case approximation ratio obtained in theory. However, this choice need not be the one leading to the best performance in practice. Therefore, we considered three versions of each algorithm, with different parameters: one was chosen empirically to obtain best possible guarantees; another is the value emanating from the theoretical analysis; and the last one is just averaging these two. We also examined the consequences of combining different algorithms. Under this scheme, all algorithms are executed in parallel and, at the end, a maximum weight matching is computed with respect to the collection of edges kept by any of these algorithms.

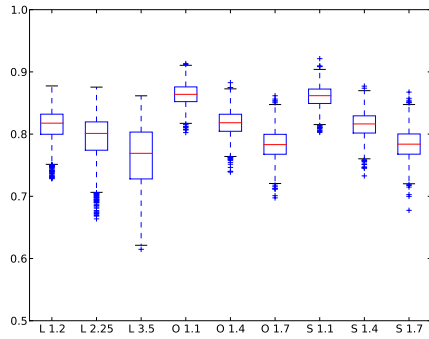
Actual tests performed. We evaluated **LAYERED**, **ONLINE**, and **SHADOW** with test graphs of roughly 1000 vertices. Following the approach of previous experimental papers in this context [13, 11], we investigated three different classes of graphs:

- **Geometric**: Points were drawn uniformly at random from the unit square; the weight of an edge is the Euclidean distance between its endpoints.
- **Real world**: Points are taken from geometric instances in the TSPLIB [15]; once again, edge weights are determined by Euclidean distances.
- **Random**: The weight of each edge is an integer picked uniformly and independently at random from $1, \dots, |V|$.

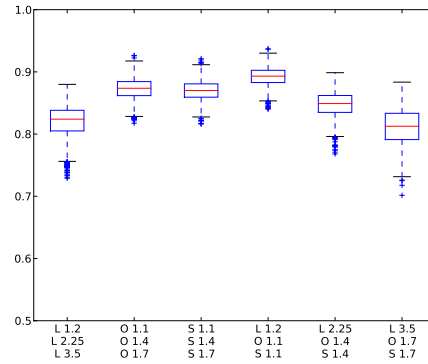
From each of these classes, we generated 10 base instances. In addition, as the performance of all algorithms under consideration depends heavily on the particular order by which edges are revealed, each algorithm was tested on every base instance for 200 independent runs, with a random edge permutation each time. To speed up the experiments all graphs were sparsified by keeping, for each vertex, the connections to one third of its closest nodes. The results are presented in Figure 2.

Conclusions. One can notice right away that the algorithms in question perform significantly better when their respective parameters are set considerably lower than the best theoretical value (1.2 for **LAYERED**, and 1.1 for **ONLINE** and **SHADOW**). With this optimization in place, it appears that **ONLINE** and **SHADOW** have comparable performance, but outperform **LAYERED**.

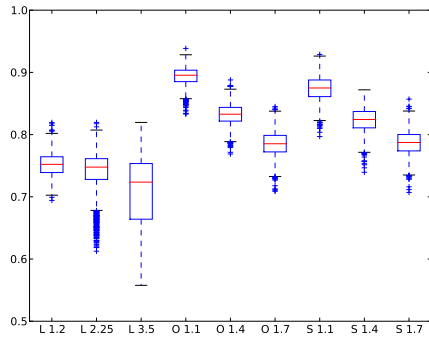
Regarding the combination of several algorithms, we compared for each algorithm the combined output of its three versions (depending on parameter setting) and the outcome of combining the best version of each of the three algorithms. We consistently observed that it is preferable to combine the output of completely different algorithms rather than the same algorithm with different parameters.



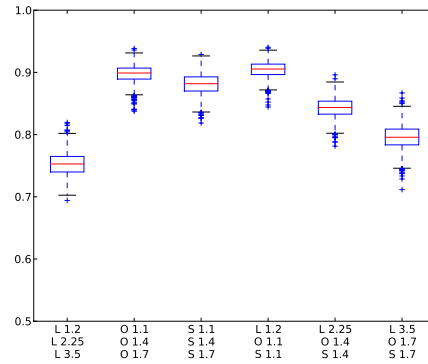
(a) Geometric. Individual algorithms.



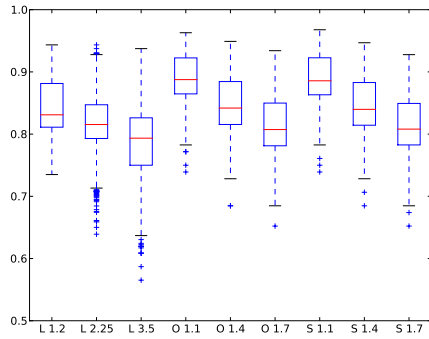
(b) Geometric. Combined algorithms.



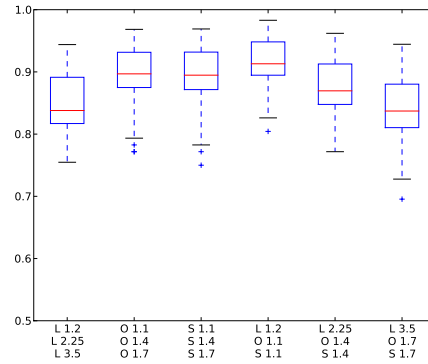
(c) Random. Individual algorithms.



(d) Random. Combined algorithms.



(e) Real. Individual algorithms.



(f) Real. Combined algorithms.

Figure 2: The results of individual algorithms appear on the left column, while the performance of combining them is shown on the right. The algorithms are specified as x-axis labels using first letters (L for LAYERED, O for ONLINE, and S for SHADOW), followed by the precise parameter value for that version. Box plots describing the outcome of our experiments are given above. Each box contains outcomes with performance between the .25 and .75 quartile, where the horizontal line inside designated the median.

Finally, we point out that, as it is often the case for approximation algorithms, the observed performance of all algorithms is significantly better than the theoretical worst case guarantee. It is worth noting, however, that their performance is still worse than traditional heuristics (such as the greedy algorithm) that are not constrained by the extent of memory usage. For example, in geometric graphs, these heuristics can recover on average 99% of the optimal value [11], whereas none of the individual algorithms can recover more than 90%.

References

- [1] N. Bansal, N. Buchbinder, A. Gupta, and J. Naor. An $O(\log^2 k)$ -competitive algorithm for metric bipartite matching. In *Proceedings of the 15th Annual European Symposium on Algorithms*, pages 522–533, 2007.
- [2] M. Elkin and J. Zhang. Efficient algorithms for constructing $(1 + \epsilon, \beta)$ -spanners in the distributed and streaming models. *Distributed Computing*, 18(5):375–385, 2006.
- [3] L. Epstein, A. Levin, J. Mestre, and D. Segev. Improved approximation guarantees for weighted matching in the semi-streaming model, 2009. <http://arxiv.org/abs/0907.0305>.
- [4] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2-3):207–216, 2005.
- [5] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. Graph distances in the data-stream model. *SIAM Journal on Computing*, 38(5):1709–1727, 2008.
- [6] J. A. Garay, I. S. Gopal, S. Kutten, Y. Mansour, and M. Yung. Efficient on-line call control algorithms. *Journal of Algorithms*, 23(1):180–194, 1997.
- [7] B. Kalyanasundaram and K. Pruhs. Online weighted matching. *Journal of Algorithms*, 14(3):478–488, 1993.
- [8] M.-Y. Kao, J. H. Reif, and S. R. Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. *Information and Computation*, 131(1):63–79, 1996.
- [9] R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 352–358, 1990.
- [10] S. Khuller, S. G. Mitchell, and V. V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theoretical Computer Science*, 127(2):255–267, 1994.
- [11] J. Maue and P. Sanders. Engineering algorithms for approximate weighted matching. In *Proceedings of the 6th International Workshop on Experimental Algorithms*, pages 242–255, 2007.
- [12] A. McGregor. Finding graph matchings in data streams. In *Proceedings of the 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 170–181, 2005.
- [13] M. Müller-Hannemann and A. Schwartz. Implementing weighted b-matching algorithms: Towards a flexible software design. *ACM Journal on Experimental Algorithmics*, 4:7, 1999.
- [14] S. Muthukrishnan. *Data Streams: Algorithms and Applications*. Foundations and Trends in Theoretical Computer Science. Now Publishers Inc, 2005.
- [15] G. Reinelt. TSPLIB. <http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/>.
- [16] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003.
- [17] M. Zelke. Weighted matching in the semi-streaming model. In *Proceedings of the 25th Annual Symposium on Theoretical Aspects of Computer Science*, pages 669–680, 2008.