Online Weighted Matching

Bala Kalyanasundaram [1] Kirk Pruhs [2]

[1]University of Pittsburgh ; Supported in part by NSF under grant CCR-9009318
[2]University of Pittsburgh

Abstract

We introduce and study online versions of weighted matching problems in metric spaces. We present a simple $2k-1$ competitive algorithm for online minimum weighted bipartite matching where $2k$ is the number of nodes. We show that this competitiveness is optimal. For online maximum matching, we prove that the greedy algorithm achieves an optimal competitive factor of 3. In contrast, we prove that the greedy algorithm performs exponentially poorly for online minimum matching.

Key words. online algorithm, matching, weighted matching, competitiveness

AMS(MOS) subject classifications. 68P05, 68Q25, 68R10, 68R05

1 Introduction

The assignment problem, finding a bipartite matching of minimum weight, is one of the archetypical problems in algorithmic graph theory and in combinatorial optimization [2, 10]. We introduce a natural online version of this problem, which we call *online min-matching*. Let G be a complete bipartite graph with one bipartition designated as the server vertices, and the other bipartition designated as the request vertices. Each bipartition has cardinality k. The weights on the edges are revealed online at k different time intervals. During the ith time interval, the weights of all the edges incident on the ith request vertex are divulged, and one unmatched server vertex must be selected to be matched with this request vertex. The goal of the online algorithm is to minimize the weight of the matching being produced. Since this is an online problem, we adopt the paradigm of competitive analysis [12]. That is, we want to minimize the worst-case ratio of the weight of the online matching to the weight of the minimum weight perfect matching. An online algorithm is *α-competitive*, or alternatively, has a *competitive factor* of α, if this ratio is at most α for all possible instances. We use the terms weight, cost and distance interchangeably throughout the paper.

Many applications of weighted matching given in introductory texts on combinatorial optimization have online versions that seem just as natural as their offline counterparts. As an example of this consider the following *fire station problem.* The server vertices represent k fire stations and the request vertices represent k fires. Each fire station can handle exactly one fire. The weight on an edge (s_i, r_j) is the distance, or equivalently time if we assume that time is proportional to distance, between fire station s_i and the location of fire r_j. The goal in the problem is to match the fire stations with the fires so as to minimize the average response time. The optimal solution is given by the minimum weight perfect matching between the fire stations and the fires. However, as we all know, life is not so convenient that all fires break out simultaneously. If fire stations must be assigned to fires as they break out, then this problem is essentially our online version of the assignment problem.

As in the fire station problem, it is common for the weights to represent distances in some metric space. So we assume that the weights are positive and satisfy the triangle inequality. It is relatively easy to see that if this assumption is not made, then there can be no algorithm for this problem with a competitive factor independent of the edge weights. Notice that the fire station problem and our online version of the assignment problem are not exactly equivalent. In the fire station problem the online algorithm has the apparent benefit of knowing the metric space and the location of the fire stations *a priori*. However, as we shall see, this additional knowledge will not provide any benefit in the worst case analysis. We should also point out that the triangle inequality implies that it never pays to have one fire station preempt another already assigned fire station. So no loss of generality occurs by not allowing preemption. Note that while r_i and r_j (s_i and s_j), with $i \neq j$, may represent the same point in the metric space, we consider them as separate vertices in the bipartite graph.

In section 2, we present the following results for online min-matching. We exhibit a metric space in which no online algorithm for min-matching can have a competitive factor less than $2k-1$. We show that the competitiveness achievable by the online algorithm is quite dependent on the underlying metric space. For some metric spaces there are algorithms for online min-matching that are arbitrarily close to 1-competitive on these metric spaces. We present a $(2k-1)$-competitive algorithm *Permutation* for online min-matching. Hence, Permutation is *strongly competitive*. That is, it gives the minimum possible competitive factor. Furthermore, Permutation does not need *a priori* knowledge of the metric space. Permutation has two other nice features, its competitiveness degrades slowly as the number of requests grows, and it can easily be extended to give a strongly competitive algorithm for the case where more than one request vertex is revealed at once.

In section 3, we consider the online version of the maximum weight bipartite matching problem [2, 10]. We call this problem *online max-matching*. Once again we must assume that the weights are nonnegative and satisfy the triangle inequality. We show that the greedy algorithm Farthest Neighbor, which services a request with the farthest available server, is 3-competitive. That is, Farthest Neighbor produces a matching of size at least 1/3 of the offline maximum weight matching. We also prove that there is no online algorithm that achieves a competitive factor less than 3. Perhaps, this is not surprising since such behavior has been shown for the offline greedy algorithm that chooses the largest available edge as the next matched edge. The offline greedy algorithm for maximum matching produces a matching of weight at least half the maximum matching [1].

For offline min-matching, Reingold and Tarjan [11] have shown that a greedy algorithm, which chooses the smallest available edge, produces a matching of weight at most a linear factor larger than the minimum matching. (Their actual result is stronger and shows that the factor is $\Theta(k^{0.58496})$.) We have already shown that a linear factor is the best possible for online min-matching. One might hope that, as in the maximum matching case, the performance of online greedy algorithm Nearest Neighbor is comparable to its offline counterpart. Unfortunately, this is not the case. We show, in section 2, that Nearest Neighbor is only $(2^k - 1)$-competitive for online min-matching, an exponential blowup from the optimal value.

Online min-matching is similar to the recently much investigated k-server problem, introduced by Manasse, McGeoch and Sleator [9]. The difference is that in the k-server problem a server handles a request instantaneously, and is available to handle the next request. This difference makes the underlying combinatorics of the two problem quite disparate. A recent survey of results for the k-server problem can be found in [3]. Our investigation into online weighted matching complements the study by Karp, Vazirani, and Vazirani [7] on online unweighted maximum cardinality matching.

2 Online Min-Matching

We begin with some basic definitions. We use R to denote the set of requested points, with r_i being the ith requested point, and R_i denoting the set $\{r_1, \ldots, r_i\}$. We use analogous notation S, and s_i for set of server points, and the ith server, respectively. We use $d(s_i, r_j)$ to represent the weight of an edge (s_i, r_j). When we refer to online min-matching on a particular metric space M, we assume that the online algorithm knows *a priori* that the metric space is M. In addition, the online algorithm knows the initial configuration of the servers before the first request is revealed. An algorithm A for online min-matching on M is α-competitive if for every possible initial configuration of the servers in M, and for every possible request sequence, the cost incurred by A is at most α times the minimum weight perfect matching.

2.1 Lower Bounds

Theorem 2.1 *For every k, there is a finite metric space M with the property that the competitive factor of every algorithm for online min-matching on M is at least $2k-1$.*

Proof: Let A be any on-line algorithm. Let M be a star, that is a tree with k leaves and $k+1$ vertices. Let all edge weights be one. The distance between any two points is the length of the shortest path. Thus the distance between two distinct leaves is 2, and the distance between a leaf and the unique nonleaf, called the root, is 1. Assume we start with one server at each leaf. The first request appears on the root. The ith request, $2 \leq i \leq k$, appears on the vertex formerly occupied by the server that A used to handle the $(i-1)$st request. The cost to A is $1+2+2+\ldots+2 = 2k-1$. The minimum matching matches the root with the one unqueried leaf for a total cost of 1. ♠

Theorem 2.1 shows that one can not achieve a sublinear competitive factor for a general metric space. However, it is possible to achieve arbitrarily low competitive factors for some fixed metric spaces.

Theorem 2.2 *For all $n > k$, and for all $\epsilon > 0$, there is a metric space M of n points such that there is a $(1+\epsilon)$-competitive algorithm for online min-matching on M.*

Proof: We construct a metric space M with n points. Let $\alpha(n)$ be a monotonically increasing function of n. Let $m_1 = 1$, and inductively define $m_i = n\alpha(n)\sum_{j=1}^{i-1} m_j$, $i \leq n$. Let the points of M be the subset $\{m_1, m_2, \ldots, m_n\}$ of the real line, with distances measured along the real line. Consider the following algorithm A for online min-matching on M. If the request is at a point occupied by an available server, then this server handles this request. Otherwise, A services each request with the leftmost unmatched/available server. (Notice that the leftmost available server may be to the right of the request point.)

We now prove by induction on number of servers k ($< n$) that the cost to A is at most $(1 + 1/(\alpha(n)))$ times the cost of the optimal offline matching. Observe that the metric space is independent of k. This clearly holds when $k = 1$. If A receives a request at a point that currently has an unmatched server, then we are done by induction. Now assume otherwise, and let (m_{a-1}, m_a) be the rightmost interval crossed by a server to serve the requests. We immediately infer that the points $m_{a+1}, \ldots, m_n$ are not requested and that initially servers do not reside on these points. We also can deduce that exactly one of the following two cases arise:

Case 1: The point m_a initially contains no servers, and $\tau > 0$ requests are made at m_a. The cost incurred by A is at most

$$\begin{aligned}
&\tau(m_a - m_{a-1}) + k\textstyle\sum_{i=2}^{a-1}(m_i - m_{i-1}). \\
&\leq \tau(m_a - m_{a-1}) + 1/\alpha(n)(m_a - m_{a-1} - n\alpha(n)) \\
&\leq \tau(m_a - m_{a-1})(1 + 1/\alpha(n)).
\end{aligned}$$

The offline algorithm must have τ servers crossing the interval (m_{a-1}, m_a) from left to right, incurring a cost of at least $\tau(m_a - m_{a-1})$.

Case 2: The point m_a initially contains $\tau > 0$ servers. Remember, we have already assumed that no requests are made at a point occupied by an available server. Hence, no requests are made at m_a since the algorithm holds these servers until the final τ requests are made. Using the same argument as in case 1, the cost to A is at most $\tau(m_a - m_{a-1})(1 + 1/\alpha(n))$. The offline algorithm must have τ servers crossing the interval (m_{a-1}, m_a) from right to left, incurring a cost of at least $\tau(m_a - m_{a-1})$. ♠

2.2 The Algorithm Permutation

We begin with some definitions and lemmas concerning optimal offline matchings. A *partial matching* of the subset R_i is a perfect matching of R_i with a subset of S. Let M_i be the set of edges that form a minimal weight partial matching on R_i with a *minimal* number of edges in $M_i - M_{i-1}$. We define M_0 to be empty. If more than one partial mapping satisfies this minimality condition, we may select one in an arbitrary, but fixed manner, and call it the *minimum weight partial matching.* Where no ambiguity will result, we also use M_i to denote the weight of this minimal matching. Let S_i be the subset of S consisting of the vertices incident to an edge in M_i.

Lemma 2.1 *The weight of the M_i's, the minimum partial matchings, form a monotonically nondecreasing sequence.*

Proof: If the weight of M_i was strictly smaller than the weight of M_{i-1} we could derive a contradiction to the definition of M_{i-1} since the edge in M_i incident to r_i has nonnegative weight. ♠

The exclusive-or, $E_1 \oplus E_2$, of two subsets of the edge set of a graph contains those edges that are in exactly one of E_1 and E_2. An alternating path (cycle) in $E_1 \oplus E_2$ is a simple path (cycle) with edges alternately in E_1 and E_2. An alternating path/cycle is *balanced* if the aggregate weight of the edges from one set equals the aggregate weight of the edges from the other set.

Lemma 2.2

a. $M_i \oplus M_{i-1}$ *consists of exactly one alternating path.*

b. For each i*, the set difference* $S_i - S_{i-1}$ *contains exactly one vertex.*

Proof: Let H be the graph formed by taking the exclusive-or of the partial matchings M_i and M_{i-1}. Vertex r_i has degree 1, each other r_j has degree 0 or 2. Each vertex in S has degree 0, 1, or 2. We can conclude from this that H consists of alternating paths and cycles. We now prove that the definitions of M_{i-1} and M_i imply that H consists of exactly one alternating path P that has r_i as an endpoint. Assume that H contains another alternating path, or cycle, $P' \neq P$. If P' was unbalanced then it would contradict either the minimality of M_{i-1}, or the minimality of M_i. If P' was balanced then it would contradict the fact that M_i was selected to minimize the number of edges in $M_i - M_{i-1}$. It is then clear that $S_i - S_{i-1}$ contains exactly one vertex. ♠

We are now ready to describe the algorithm Permutation. Let P_i denote the partial matching constructed by Permutation after the first i requests. Permutation gets its name because it maintains the following invariant:

> *For all* i*, the vertices in* S *incident to an edge in* M_i *are exactly the vertices in* S *that are incident to an edge in* P_i*.*

If a request is on a location occupied by an unused server, then Permutation matches that server to the request. Otherwise, Permutation responds to the $(i+1)$st request by computing M_{i+1} and servicing r_{i+1} with the unique vertex s in S that is in $M_{i+1} - M_i$.

Theorem 2.3 *Permutation is* $2k-1$*-competitive.*

Proof: We prove inductively that the weight of P_i is at most $2i-1$ times the weight of M_i. For i equal to 1, $M_i = P_i$ and the result follows. Now assume that the result holds for $i-1$. Assume P services r_i with s_j. Let M' be the union of the matching M_{i-1} and (s_j, r_i) and let H be the exclusive-or of M_i and M'. H consists of one alternating cycle. By the triangle inequality we have that $d(s_j, r_i)$ is at most the aggregate weight of the edges in H minus $d(s_j, r_i)$, which is at most $M_{i-1} + M_i$. Hence, $P_i - P_{i-1} \leq M_i + M_{i-1}$. Using the fact that $M_{i-1} \leq M_i$ and the inductive hypothesis we have

$$P_i \leq (2(i-1)-1)M_{i-1} + 2M_i \leq (2i-1)M_i$$

$\spadesuit$

Note that by remembering M_{i-1}, Permutation can compute M_i, and thus handle a request, in time that it takes to solve one single source shortest path problem in a graph with $k+i$ vertices and ki edges. This problem can be solved in time $O(ki + k\log n)$ [4].

If some of the k requests arrive simultaneously we can modify Permutation to obtain a better competitive factor. More precisely, if the requests come in t groups we obtain a $(2t-1)$-competitive algorithm, which is optimal. This algorithm is based on the following lemma, whose proof is similar to the proof of lemma 2.2.

Lemma 2.3
a. The exclusive-or of M_i and M_{i-j+1} consists of exactly j disjoint alternating paths where each starts with a request point in $\{r_{i-j+1}, r_{i-j+2}, \ldots, r_i\}$.
b. For each $i > j$, the set difference $S_i - S_{i-j+1}$ contains exactly j vertices.

Assume that the ith group has j requests. Let H be the exclusive-or of the new minimum weight partial matching and the previous minimum weight partial matching. The online algorithm matches these j requests with the corresponding endpoints of the j alternating paths in H. More precisely, if the alternating path starts at r_l and ends in s_m then r_l and s_m are matched.

Theorem 2.4 *This variant of Permutation is* $(2t-1)$*-competitive for online min-matching if the requests arrive in* t *groups.*

Proof: The proof closely parallels the proof of theorem 2.3. If requests $i+1$ through $i+j$ arrive simultaneously then one can derive the inequality $P_{i+j} - P_i \leq M_{i+j} + M_i$. There are t such inequalities. If we sum up the left hand side we get P_k, whereas the right hand side is at most $(2t-1)M_k$. Note that $P_0 = M_0 = 0$. ♠

2.3 Analysis of Nearest Neighbor

We now analyze the competitiveness of the greedy algorithm, Nearest Neighbor. Nearest Neighbor chooses the closest available server to handle the current request, breaking ties arbitrarily. We show that Nearest Neighbor is $(2^k - 1)$-competitive and that this bound is tight.

Theorem 2.5 *Nearest Neighbor achieves a competitive factor of* $2^k - 1$.

Proof: Let N_i be the partial matching constructed by Nearest Neighbor after i vertices have been revealed. We continue the convention of also using N_i to refer to the weight of the ith partial mapping constructed by Nearest Neighbor. We can assume, without loss of generality, that by renumbering the vertices that Nearest Neighbor services r_i with s_i. We prove inductively that $N_i \leq (2^i - 1)M_i$. For $i = 1$, $M_i = N_i$ and the result follows. Now assume that the result holds for $i-1$ and we verify it holds for i. Assume that r_i is matched with s_j in M_i.

If the weight of the edge selected by Nearest Neighbor to service r_i is at most the weight of the edge incident to r_i in M_i then we are done. Otherwise, it must be the case that s_j is incident on an edge in N_{i-1}. Let H be the exclusive-or of N_{i-1} and M_i. There is a unique maximal alternating path P starting from r_i and ending at a vertex s_l not incident to any edge in N_{i-1}. Then $d(r_i, s_i) \leq d(r_i, s_l)$ by the definition of Nearest Neighbor. Also $d(r_i, s_l) \leq M_i + N_{i-1}$ by the triangle inequality. Since $N_i = N_{i-1} + d(s_i, r_i)$, $N_i \leq N_{i-1} + d(r_i, s_l) \leq 2N_{i-1} + M_i$. Then by the inductive hypothesis $N_i \leq 2(2^{i-1} - 1)M_{i-1} + M_i$. By lemma 2.1, we know that $M_{i-1} \leq M_i$. Hence we can conclude $N_i \leq (2^i - 1)M_i$. ♠

Theorem 2.6 *For all k there is a finite metric space M for which Nearest Neighbor has a competitive factor of* $2^k - 1$*.*

Proof: M is a subspace of the real line with distances being defined in the standard way. Let $s_1 = -1$, and $s_i = 2^{i-1} - 1$, $2 \leq i \leq k$. The request points are $r_i = 2^{i-1} - 1$, $1 \leq i \leq k$. We can assume Nearest Neighbor uses s_{i+1} to service request r_i for $i < k$ and services r_k with s_1. The cost to Nearest Neighbor is $2^k - 1$. The optimal matching is obtained by matching each r_i to the corresponding s_i for a total cost of 1. ♠

3 Maximum Weighted Matching

In this section we show that the greedy algorithm, Farthest Neighbor, which chooses the farthest available server to handle the current request, is 3-competitive, and that this bound is tight.

Theorem 3.1 *Farthest Neighbor produces a matching of weight at least* $1/3$ *the maximum weight perfect matching.*

Proof : Let s_i be the server chosen by Farthest Neighbor to match r_i. For all $j > i$, $d(s_j, r_i) \leq d(s_i, r_i)$, otherwise Farthest Neighbor would have chosen s_j instead of s_i. Now consider the optimal matching in which the request r_i is served by $s_{\pi(i)}$. We prove that if $i > \pi(i)$ then $D(r_i) = d(s_{\pi(i)}, r_i) - d(s_i, r_i)$ is at most $2 * d(s_{\pi(i)}, r_{\pi(i)})$. Also, if $i \leq \pi(i)$ then $D(r_i) \leq 0$. Therefore, $\sum_{i=1}^{n} D(r_i) \leq \sum_{i=1}^{n} 2 * d(s_{\pi(i)}, r_{\pi(i)})$. Since $i \neq j$ implies $\pi(i) \neq \pi(j)$, the right hand side of the above inequality is two times the matching induced by Farthest Neighbor. The left hand side is the difference between optimal matching and the matching induced by the Farthest Neighbor. Therefore, optimal matching is at most three times the matching induced by Farthest Neighbor.

Therefore, we are left with one inequality to prove that if $i > \pi(i)$, then $D(r_i) \leq 2 * d(s_{\pi(i)}, r_{\pi(i)})$. We use the triangle inequality to prove this. We know (by Farthest Neighbor) $d(s_{\pi(i)}, r_{\pi(i)}) \geq d(s_i, r_{\pi(i)})$. By triangle inequality, $d(s_{\pi(i)}, s_i) \leq 2 * d(s_{\pi(i)}, r_{\pi(i)})$ and $D(r_i) \leq d(s_{\pi(i)}, s_i)$. ♠

We prove that Farthest Neighbor is optimal by showing that there is no algorithm that achieves a competitive factor strictly less than 3 for every metric space.

Theorem 3.2 *There is a metric space space M with the property that no online algorithm for online max-matching on M can guarantee a matching larger than $1/3$ times the maximum weight perfect matching.*

Proof: For each i, $1 \leq i \leq k$, we have two points a_i and b_i in M. Additionally there a center point c. The distance between each a_i and each b_i is 3. The distance between each a_i and c is 1. The distance between an a_i and a b_j, $i \neq j$, is 1. All other distances can be computed by taking shortest paths.

One can easily verify that M satisfies the triangle inequality. Figure 3.1 contains a representation of M for $k = 3$. In this figure darkened edges have weight 3, and light edges have weight 1.

The servers are initially placed at the a_i's. The first request is to c. Assume that the server at $a_{\gamma(j)}$ handled the jth request, $1 \leq j \leq k-1$. Then the $(j+1)$st request is to the vertex $b_{\gamma(j)}$ if $j \leq k-2$. The last request is at the location of the one unmatched server. The cost to the online algorithm is then $k-1$. The optimal offline matching is obtained by matching the server initially at $a_{\gamma(j)}$ with $b_{\gamma(j)}$ for $1 \leq j \leq k-2$, the server initially at $a_{\gamma(k-1)}$ with the last request, and the server at the position of the last request with c, for a total cost of $3(k-2) + 2 + 1 = 3(k-1)$.

Figure 3.1

♠

4 Conclusion and Open Problems

This paper reports on preliminary results from our ongoing investigation into online matching problems. This area is filled with many interesting open problems. We conclude by presenting several of these problems.

One common variant of weighted matching is *bottleneck matching* [10]. In bottleneck matching, the cost of a matching is the weight of the largest edge and the goal is to find the matching with minimum bottleneck weight. The goal is still to minimize the ratio of the bottleneck weight of the online matching to the weight of the minimum bottleneck matching. The combinatorics of online bottleneck matching seem to be somewhat different from those of regular bipartite matching. Preliminary results for this problem have been reported by Idury and Schaffer [5].

One might also consider online transportation problems [2, 10]. A transportation is generalization of the assignment problem in that each server vertex is allowed to handle an arbitrary, but fixed, number of requests. For example, in the transportation version of the fire station problem each fire station may have several fire crews, and each fire crew can handle one fire. The goal would be to determine the competitive factor as a function of the number of fire stations instead of the number of servers.

Randomization has been shown to be very helpful in designing online algorithms. However, harnessing the full power of randomization is often not easy. It would be interesting to study randomized algorithms for online min-matching.

The conference version of this paper [6] contains some results comparing the optimal competitive factor for the k-server problem to the optimal competitive factor for online min-matching. In between these two extreme problems lie a range of server problems in which the time required to service a request is between zero and infinity. This spectrum of problems seems worthy of further investigation.

Acknowledgements: It has recently been brought to our attention that some of the results presented in this paper for online min-matching were obtained independently by Khuller, Mitchell and Vazirani [8].

References

[1] D. Avis, *A survey of heuristics for the weighted matching problem*, Networks, 13 (1983), pp. 475–493, 1983.

[2] B. Carre, *Graphs and Networks*, Clarendon Press, 1979.

[3] A. Fiat, Y. Rabani, and Y. Ravid, *Competitive k-server algorithms*, Proceedings of the 31st IEEE Symposium on Foundations of Computer Science, 1990, pp. 454–463.

[4] M. Fredman, and R. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, J. ACM, 34 (1877), pp. 596–615.

[5] R. Idury, and A. Schäffer, *A better lower bound for online bottleneck matching*, manuscript.

[6] B. Kalyanasundaram, and K. Pruhs *Online Weighted Matching*, Proceedings of the 2nd SIAM Conference on Discrete Algorithms, 1991, pp. 234–240.

[7] R. Karp, U. Vazirani, and V. Vazirani, *An optimal algorithm for online bipartite matching*, Proceedings of the 22nd ACM Symposium on Theory of Computing, 1990, pp. 352–358.

[8] S. Khuller, S. Mitchell, and V. Vazirani, *Online algorithms for weighted matching and stable marriages*, Tech. Report 90-1143, Department of Computer Science, Cornell University, 1990.

[9] M. Manasse, L. McGeoch, and D. Sleator, *Competitive algorithms for online problems*, Proceedings of the 20th ACM Symposium on Theory of Computing, 1988, pp. 322–333.

[10] C. Papadimitriou and K. Steiglitz, *Combinatorial Optimization*, Prentice-Hall, 1982.

[11] E. Reingold and R. Tarjan, *On a greedy heuristic for complete matching*, SIAM J. Comp., 10 (1981), pp. 676–681.

[12] D. Sleator and R. Tarjan, *Amortized efficiency of list update and paging rules*, Communications of the ACM, 28 (1985), pp. 202–208.