Online matching with blocked input

By: Ming-Yang Kao and Stephen R. Tate

M. Kao and S. R. Tate. “Online Matching with Blocked Input”, Information Processing Letters, Vol. 38, May 1991, pp. 113–116.

Abstract:

In this paper, we examine the problem of "blocked online bipartite matching". This problem is similar to the online matching problem except that the vertices arrive in blocks instead of one at a time. Previously studied problems exist as special cases of this problem; the case where each block contains only a single vertex is the standard online matching problem studied by Karp et al. (1990), and the case where Mere is only one block (containing al/ vertices of the graph) is the offline matching problem (see, for example, the work by Aho et al. (1985)).

The main result of this paper is that no performance gain (except in low-order terms) is possible by revealing the vertices in blocks, unless the number of blocks remains constant as n (the number of vertices) grows. Specifically, we show that if the number of vertices in a block is $k = o(n)$, then the expected size of the matching produced by any algorithm (on its worst-case input) is at most $(1 — 1/e)n + o(n)$. This is exactly the bound achieved in the original online matching problem, so no improvement is possible when $k = o(n)$. This result follows from a more general upper bound that applies for all $k \leq n$; however, the bound does not appear to be tight for some values of k which are a constant fraction of n (in particular, for $k = n/3$).

We also give an algorithm that makes use of the blocked structure of the input. On inputs with $k = o(n)$, this algorithm can be shown to perform at least as well as using the algorithm from Karp et al. (1990) and ignoring blocking. Hence, by the upper bound, our algorithm is optimal to low-order terms for $k = o(n)$, and in some cases considerably outperforms the algorithm of Karp et al. (1990). The algorithm also trivially has optimal performance for $k = n$; furthermore, it appears to have optimal performance for $k = n/2$, but a proof of this performance has not been found. Unfortunately, the algorithm does not meet the upper bound for all block sizes, as is shown by a simple example with block size $n/3$. We conjecture that the algorithm we present is actually optimal, and that the upper bound is not tight.

Keywords: Analysis of algorithms, online algorithms

Article:

1. Introduction

In this paper, we examine the problem of *blocked online bipartite matching*, which is a generalized problem that includes both classical (off-line) matching and online matching. The input for the blocked online matching problem[1] is a bipartite graph with a perfect matching; the vertices are revealed in small blocks. To distinguish the halves of the bipartite graph, we give each half a color: either red or blue. The input graph is assumed to have $2n$ vertices (n red vertices and n blue vertices), and the red vertices are presented k at a time, with all their adjacencies. When each block of k vertices is revealed, the algorithm must decide what matches to make involving these vertices, and the matches involving this block can never be changed after the block is processed. The input graph is chosen at the beginning of the algorithm; in the language of online algorithms, this is the same as assuming an oblivious adversary—the only difference is that the size of the input is fixed (and known by the online algorithm) at the beginning of the computation.

The blocked online matching problem contains as special cases the standard online problem (with $k = 1$) as studied by Karp, Vazirani, and Vazirani [2], and the classical, offline matching problem (with $k = n$), as well as the entire spectrum between these two previously studied problems. The *performance* of these algorithms is measured by the expected number of vertices matched on the worst-case input for the online algorithm. Clearly, with $k = n$, a matching of size n can easily be found (remember that the input graph is guaranteed to contain a perfect matching), so performance of n can be achieved. The case of $k = 1$ was studied in [2], and they proved a performance upper bound of $(1 - 1/e)n + o(n)$ for any algorithm, in addition to presenting an algorithm that matched this bound up to low-order terms.

In Section 2 we prove an upper bound that relates the blocked matching problem to the non-blocked online matching problem. This bound shows, among other things, that for all $k = o(n)$, the best performance possible is $(1 - 1/e)n + o(n)$, so nothing is gained (except in low-order terms) by blocking the input. An algorithm is presented in Section 3 that gives optimal performance for $k = o(n)$ and in the extreme case of $k = n$. Furthermore, the upper bound shows that no algorithm can obtain performance better than $(3/4)n$ when $k = n/2$, and the algorithm of Section 3 appears to match this bound. Unfortunately, the performance of the algorithm does not match the upper bound for smaller $k = \Theta(n)$. We conjecture that the algorithm obtains optimal performance and that the upper bound is not tight.

2. The upper bound

In this section, we prove an upper bound on the performance of any algorithm that solves the blocked online matching problem. The upper bound is not stated in terms of a function of n; rather, it is stated in terms of the optimal performance of any nonblocked online matching algorithm on smaller graphs.

Let $\mathcal{A}(k, n)$ be the class of all (randomized) online algorithms that construct a matching on 2n-vertex bipartite graphs (each input is guaranteed to have a perfect matching), where the red vertices are revealed in blocks of k vertices each. Let $\mathcal{B}(k, n)$ be the set of all possible inputs to an algorithm in $\mathcal{A}(k, n)$; in other words, $\mathcal{B}(k, n)$ is the set of all $2n$-vertex bipartite graphs that contain a perfect matching, where the ordering of the vertices is important. $EM(k, n)$ is defined to be the expected size of the matching produced by the best algorithm from $\mathcal{A}(k, n)$ on its worst-case input. Specifically, if $M_A(x)$ is a random variable denoting the size of a matching produced by algorithm $A \in \mathcal{A}(k, n)$ on input $x \in \mathcal{B}(k, n)$, then $E[M_A(x)]$ is the *performance* of A on x, and

$$EM(k, n) = \max_{A \in \mathcal{A}(k,n)} \min_{x \in \mathcal{B}(k,n)} E\left[M_A(x)\right].$$

We call an algorithm in $\mathcal{A}(k, n)$ optimal if it has worst-case performance equal to $EM(k, n)$. We note here that any optimal algorithm *must* use randomization. This follows from an extension of the deterministic upper bound in [2]; specifically, no deterministic algorithm can have performance better than $n/2$ when $k \leq n/2$. Notice that $EM(1, n)$ is exactly the best worst-case performance of a nonblocked online algorithm, as examined in [2]. We relate performance of blocked and nonblocked online algorithms in the following theorem:

Theorem 2.1.

$$EM(1, kn) \leqslant EM(k, kn) \leqslant k \cdot EM(1, n).$$

Proof. The first inequality, $EM(1, kn) \leq EM(k, kn)$, is obvious. The blocking of the input can be ignored, and the graph matched one vertex at a time using an optimal algorithm for non-blocked online matching.

The second inequality is proved by contradiction. Assume that there exists a blocked online algorithm $A \in \mathcal{A}(k, kn)$ with worst-case performance $> k \cdot EM(1, n)$. We can assign an order to the vertices in each block, so the notion of the ith vertex $(1 \leq i \leq k)$ in a block is well defined. For a given input x, let $E[M_{A,i}(x)]$ denote the expected number of ith position vertices matched (in all blocks); notice that this means that

$$E[M_A(x)] = E[M_{A,1}(x)] + E[M_{A,2}(x)] + \cdots + E[M_{A,k}(x)].$$

Now consider the following nonblocked algorithm *S*. First, pick a value *c* uniformly from {1, 2, ... , *k*}. Next, when a vertex is presented to the nonblocked algorithm *S*, make *k* distinct copies of the vertex (and its adjacencies), and simulate algorithm *A* on this "block". Finally, the match that *S* makes on this step is the match that *A* made on the *c*th vertex in the block. The simulation steps are reapplied for each vertex received by *S*, each time using the original value of *c* and making *k* copies of the original graph.

Since algorithm *S* has constructed *k* copies of the input graph, and we have randomly chosen a copy and taken its matching, the result is clearly a valid matching in the original graph. Let *x* denote the input to *S*, and let $\bar{x}$ denote the graph constructed for input to algorithm *A* (i.e., $\bar{x}$ is the graph with *k* distinct copies of *x*). The performance of algorithm *S* on input *x* is

$$E[M_S(x)] = \frac{1}{k}E[M_{A,1}(\bar{x})] + \frac{1}{k}E[M_{A,2}(\bar{x})] + \cdots + \frac{1}{k}E[M_{A,k}(\bar{x})] = \frac{1}{k}E[M_A(\bar{x})].$$

The algorithm *A* was chosen to be such that for all $\bar{x}$, $E[M_A(\bar{x})] > k \cdot EM(1, n)$, so for all *x*, $E[M_S(x)] > EM(1, n)$. Since *EM(1, n)* is the optimal worst-case performance, this shows that the performance of *S* is better than optimal—clearly a contradiction, completing the proof of the theorem.

The following theorem shows that blocking input has no benefit (to low-order terms) when $k = o(n)$:

Theorem 2.2. *If N = kn and k = o(N), then*

$$\lim_{N\to\infty} \frac{EM(k, N)}{N} = \lim_{N\to\infty} \frac{EM(1, N)}{N}.$$

Proof. Rewriting the bound from Theorem 2.1 gives

$$\frac{EM(1, N)}{N} \leqslant \frac{EM(k, N)}{N} \leqslant \frac{EM(1, N/k)}{N/k}.$$

Since *k = o(N), N/k* must grow without bound as *N* increases. This gives

$$\lim_{N\to\infty} \frac{EM(1, N)}{N} = \lim_{N\to\infty} \frac{EM(1, N/k)}{N/k}.$$

Furthermore, since *EM(k, N)/N* is sandwiched between these two functions, its limit must also be the same, which is exactly the claim in the theorem.

Using a result from [2], an exact value can be given for *EM(k, N)*, when *k = o(N).*

Corollary 2.3. *If N = kn and k = o(N), then*

$$EM(k, N) = \left(1 - \frac{1}{e}\right)N + o(N).$$

EM(1, *n*) is easy to compute for small values of *n.* Table 1 lists *EM*(1, *n*) for $n \leq 5$; note that these numbers provide an upper bound on *EM(n/c, n)* for $c \leq 5$.

Table 1
EM(1, *n*) for small *n*

n	*EM*(1, *n*)
1	1
2	3/2
3	13/6
4	67/24
5	411/120

3. The blocked input algorithm

In this section, we present an algorithm for the blocked online matching problem. The algorithm has several interesting features. First, when $k = 1$, the algorithm is exactly the same as the online matching algorithm of Karp, Vazirani, and Vazirani [2]. They conjectured that their online algorithm was absolutely optimal (they proved that their algorithm is optimal up to low-order terms, and the conjecture states that their algorithm has optimal performance even when low-order terms *are* considered), so if this is true, our algorithm is also optimal for $k = 1$. Secondly, for $k = n$, the algorithm always gives a matching of size n, so it is trivially optimal in that case. Finally, this algorithm is optimal to low-order terms for $k = o(n)$.

The first step of the algorithm is to pick a permutation a of the integers $(1, 2, \ldots, n)$ uniformly from all possible permutations. This permutation will remain fixed throughout all blocks of input, and should be viewed as ranking the blue vertices. The algorithm should find a maximum size matching in every block it receives, giving priority to the highest ranked blue vertices; in other words, the algorithm chooses the lexicographically first maximum matching. This can be accomplished by assigning weights to the edges of a block as follows: for any edge $e = (u_i, v_j)$, where u_i is a red vertex and v_j is a blue vertex, assign a weight

$$w(e) = 2^{n+1} + 2^{\sigma(j)}.$$

After all edges have been assigned a weight, a maximum weight matching is found. The term 2'1 in each edge's weight insures that the matching involves the maximum number of vertices, arid the remaining term in the weight gives priority to the highest-ranked set of blue vertices that are in a maximum matching. It is not hard to see that this algorithm becomes the same as the [2] algorithm when $k = 1$.

For $k = o(n)$, the algorithm performs *at least* as well as running [2] and ignoring the blocking. Therefore, this algorithm is optimal to low-order terms when $k = o(n)$. The expected size of the matching produced when $k = n/2$ seems to be $(3/4)n$, which is the best possible by Theorem 2.1 and Table 1; unfortunately, a proof of this performance has not been found.

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Fig. 1. Bad case for $k = n/3$.

For $k = n/3$, the story becomes worse. For the adjacency matrix of Fig. 1 (where the columns are revealed two at a time from the right), the expected matching size for the above algorithm was computed by a program that counted matchings over all permutations of the rows; the expected size of the matching produced by the algorithm of this section is $377/90 \approx 4.19$, which is slightly less than the upper bound of $13/3 \approx 4.33$ provided by Theorem 2.1.

4. Conclusion

It has been shown in this paper that unless a constant fraction of the total number of vertices are revealed in a block, then blocking the input has no significant advantage over simply revealing the vertices one at a time. When $k = \Theta(n)$, blocking the vertices can be of considerable advantage.

We have also presented an algorithm that takes advantage of the blocked structure of the input. Due to the upper bound, this gives no asymptotic gain over [2] for $k = o(n)$, but seems to perform well for large block sizes where [2] becomes less than optimal. Analysis of the expected matching size produced by the algorithm has proved to be a very difficult problem, which we leave open for the present. We believe that the algorithm is optimal, and that a tighter upper bound is possible.

Notes:

[1] To avoid being too wordy, the word "bipartite" is left out of most of this discussion; however. it should be remembered that we are dealing *only* with bipartite graphs.

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *Data Structures and Algorithms* (Addison-Wesley, Reading, MA, 1985).

[2] R.M. Karp, U.V. Vazirani and V.V. Vazirani, An optimal algorithm for on-line bipartite matching, in: *Proc. 22nd STOC* (1990) 352-358.