This paper appeared in: Information Processing Letters, 99:3 (2006), 119-123

Approximating Weighted Matchings in Parallel

STEFAN HOUGARDY & DORATHA E. VINKEMEIER

Humboldt-Universität zu Berlin
Institut für Informatik
10099 Berlin, GERMANY
{hougardy,drake}@informatik.hu-berlin.de

revised Version

Abstract. We present an NC approximation algorithm for the weighted matching problem in graphs with an approximation ratio of $(1-\epsilon)$. This improves the previously best approximation ratio of $(\frac{1}{2}-\epsilon)$ of an NC algorithm for this problem.

Keywords. approximation algorithms, parallel algorithms, analysis of algorithms, graph algorithms, maximum weight matching

1 Introduction

The class NC is the class of all problems that are computable in polylogarithmic time with polynomially many processors. A major open problem in parallel algorithms is the question whether there exists an NC algorithm for the maximum cardinality matching problem in graphs. There exist $O(\sqrt{|V|}|E|)$ sequential algorithms for this problem [9, 13], but no NC algorithm is known, not even for the special case of bipartite graphs. Karp, Upfal and Wigderson [8] have shown that there exists an RNC algorithm for the maximum cardinality matching problem. Mulmuley, Vazirani and Vazirani [10] presented another such algorithm with improved running time. Karloff [6] established that the maximum cardinality matching problem belongs to ZNC.

This paper appeared in: Information Processing Letters, 99:3 (2006), 119-123

Fischer, Goldberg, Haglin and Plotkin [3] have shown that there exists an NC-approximation scheme for the maximum cardinality matching problem.

In this paper we are interested in parallel algorithms for the *weighted* matching problem, i.e., the problem of finding a matching of maximum weight in an edge weighted graph. The maximum cardinality matching problem is a special case of the weighted matching problem where all edges have weight one. Therefore, there are also no NC algorithms known for finding an optimal solution to the weighted matching problem. This problem is also not known to belong to RNC. In the special case that the edge weights are given in unary RNC algorithms are known [8, 10]. Uehara and Chen [12] have shown that there exists an NC approximation algorithm for the weighted matching problem that achieves an approximation ratio of $\frac{1}{2}-\epsilon$. In this paper we improve this result by presenting a $1-\epsilon$ NC approximation algorithm for the weighted matching problem.

Our algorithm makes use of a combination of the ideas from the NC-approximation scheme for the maximum cardinality matching problem due to Fischer, Goldberg, Haglin and Plotkin [3], from the $\frac{1}{2}-\epsilon$ approximation algorithm of Uehara and Chen [12], and from a recent linear time sequential $\frac{2}{3}-\epsilon$ approximation algorithm due to Drake and Hougardy [1, 2].

2 Preliminaries

A *matching* M in a graph $G = (V, E)$ is a subset of the edges E of G such that no two edges in M have a vertex in common. The maximum cardinality matching problem is to find a matching of maximum cardinality in a graph. Let $G = (V, E)$ be a graph and $w : E \to \mathbb{R}_+$ be a function which assigns a positive weight to each of the edges of G. Then the weight $w(F)$ of a subset $F \subseteq E$ of the edges of G is defined as $w(F) := \sum_{e \in F} w(e)$. The weighted matching problem is to find a matching M in G that has maximum weight.

The model of computation used in this paper is the CREW PRAM (concurrent reads exclusive writes parallel random access machine). In this model there exists a sequence of indexed random access machines, each of which knows its own index. The processors synchronously execute the same central program, communicating with one another through a shared random access memory. CREW means that several processors can concurrently read a particular memory address but at most one processor can write to a single memory address in each step. See [5, 7] for more background on parallel algorithms.

The quality of an approximation algorithm for the weighted matching

This paper appeared in: Information Processing Letters, 99:3 (2006), 119-123

problem is measured by its so-called *approximation ratio*. An approximation algorithm has an approximation ratio of c, if for all graphs it finds a matching with a weight of at least c times the weight of an optimal solution.

3 The Ranked Augmentation Graph

The main idea of our algorithm is to start with some (possibly empty) matching and to allow each processor to make some local changes of this matching to improve its weight. The local changes made by different processors need to be independent of each other. To achieve this we construct from the given graph a new graph which we call the augmentation graph. In this augmentation graph an independent set of vertices corresponds to a set of pairwise independent local changes of the matching in the given graph. As we aim to increase the weight of a given matching by a set of local changes by as much as possible, we will *rank* all possible local changes for a given matching. This means that we partition the set of all possible local changes of a matching into classes such that the local changes within each class achieve a similar increase in weight. Instead of computing an independent set in the augmentation graph the idea is to compute an independent set in each of the classes of the partition. The augmentation graph together with the ranking of its vertices will be called the ranked augmentation graph. We are going to describe the construction of this graph in more detail in the following.

Given a graph $G = (V, E)$ and a matching $M \subseteq E$ let an *augmentation with respect to the matching* M be any matching $S \subset E \setminus M$. If the matching M is known from the context we will simply say that S is an augmentation. Let $M(S)$ denote all edges in M that have a vertex in common with an edge in S. Then $(M \setminus M(S)) \cup S$ is again a matching. We say that $(M \setminus M(S)) \cup S$ is the matching that is obtained by augmenting M by S. If S is an augmentation with respect to M then the *gain* of augmenting M by S is defined as $gain_M(S) = w(S) - w(M(S))$. Thus the gain is the difference of weight between the matching M and the matching $(M \setminus M(S)) \cup S$. Our definition allows augmentations that have negative gain, but our algorithm will only consider augmentations that have positive gain, i.e., that increase the weight of the matching M.

The *size* of an augmentation S is simply the number of edges contained in S. Let $G = (V, E)$ be a graph, $M \subseteq E$ be a matching and $l > 0$ be an integer. Then the *augmentation graph* $G' = G'(G, M, l)$ is defined as follows. The vertices of G' are all augmentations with respect to M of size

This paper appeared in: Information Processing Letters, 99:3 (2006), 119-123

at most l. Two such vertices are connected by an edge if the corresponding augmentations have at least one vertex of G in common.

All augmentations which are vertices of G' will be ranked according to their gains as follows (this is very similar to the ranking of the edges that Uehara and Chen [12] used in their algorithm). First find the augmentation of $V(G')$ with the largest gain denoted by $gain_{max}$. For each vertex S in $V(G')$ we define its *rank* $r(S)$ as follows (n denotes the number of vertices in G)

- If $gain(S) \leq \frac{gain_{max}}{l \cdot n}$ then $r(S) = 0$
- Otherwise $r(s) = i > 0$ where i is the smallest integer for which it is true that $gain(S) \leq 2^i \cdot \frac{gain_{max}}{l \cdot n}$.

This definition implies that for constant l the rank of an augmentation is an integer of size $O(\log n)$.

We call the augmentation graph G' together with the ranking of its vertices the *ranked augmentation graph* of G. This graph can be computed using $O(n^{4l})$ processors and $O(\log n)$ time as follows. Use $O(n^{2l})$ processors to generate all possible augmenting sets S of size at most l. For each such set the assigned processor can check in constant time whether it is a matching. Next remove all sets that do not form a matching. This is possible in $O(\log n)$ time using list compression methods (see for example [5]). Now we have generated all vertices of G'. Assign one processor to each of the $O(n^{4l})$ pairs of vertices to check in constant time whether the corresponding augmenting sets intersect.

4 The Algorithm

Our algorithm for computing a $1 - \epsilon$ approximation of a maximum weight matching starts with the empty matching and makes c calls to the algorithm `ImproveMatching` for some constant c which depends only on ϵ. The algorithm `ImproveMatching` is shown in Figure 1. This algorithm takes as input a weighted graph G and a matching M and returns a new matching M'. It starts by calculating out of G and M the ranked augmentation graph G' as described in Section 3. Within the graph G' a maximal independent set is calculated in the following way. Let V_i be the vertices of G' that have rank i. Then starting from the highest rank a maximal independent set ALG_i is calculated in the subgraph of G' that is induced by the vertices of V_i which have not yet been removed from G'. All neighbors of vertices of ALG_i are

This paper appeared in: Information Processing Letters, 99:3 (2006), 119-123

`ImproveMatching`: $G = (V, E), w : E \to R_+$, matching M
Output: matching M'

ALG $= \emptyset$
calculate the ranked augmentation graph G'
for $i = rank_{max}$ **downto** 1 **do**
  calculate a maximal independent set ALG_i in the graph $G'_i := (V'_i, E'_i)$ that is induced by all vertices still in G' that have rank i
  remove all vertices from G' that have neighbors in ALG_i
  $ALG = ALG \cup ALG_i$
$M' = M$ augmented by all augmentations in ALG

Figure 1: An NC algorithm for improving the weight of a matching M.

removed to ensure that the union of all sets ALG_i is an independent set of G'. The process considers all vertices from the highest rank down to those of rank 1. Vertices of rank 0 are thrown away. The set ALG is the union of all the sets ALG_i and is by construction a maximal independent set in $G' \setminus V_0$. ALG is used as an augmenting set for M to obtain the new matching M' which is returned by the algorithm `ImproveMatching`.

5 The Analysis of the Algorithm `ImproveMatching`

For the analysis of the algorithm `ImproveMatching` we will define a multiset OPT of vertices of G'. The idea of the set OPT is that it can be seen as a fractional covering of augmentations that all together yield a maximum weight matching in G. Let M^* be a maximum weight matching in G. Consider the symmetric difference of M and M^*. This consists of alternating paths and cycles. Let C denote a cycle or path in this symmetric difference and let C^* denote the edges $C \cap M^*$. If $|C^*| \leq l$ then put the augmentation C^* (vertex of G') in OPT with multiplicity l. If $|C^*| > l$ (is long) then there are two cases: either C is a cycle or C is a path. If C is a long cycle then for each edge e of C^* add to OPT the augmentation that begins with e and includes the next $l - 1$ edges of C^* as they occur consecutively in C. This way there are $|C^*|$ different augmenting sets defined over C each containing l edges of M^*. If C is a long path then consider the edges of C^* to be consecutively indexed from 1 to $p := |C^*|$ and use wrap around so that the lowest index edge e_1 follows the highest indexed edge e_p as for a long cycle and add to OPT the same augmenting sets for C as for a long cycle.

By definition of OPT the edges of $M^* \setminus M$ are contained in at least l augmentations from OPT. Moreover, for each edge of $M \setminus M^*$ there exist

This paper appeared in: Information Processing Letters, 99:3 (2006), 119-123

at most $l+1$ augmentations $S \in OPT$ such that the edge is contained in $M(S)$. Therefore we have:

$$\sum_{S \in OPT} gain(S) \;\geq\; l \cdot w(M^*) - (l+1) \cdot w(M) \tag{1}$$

According to how we defined the multiset OPT the number of augmentations in OPT is bounded by $l \cdot n$. Vertices in V_0 have rank 0 and the corresponding augmentations a gain of at most $\frac{gain_{max}}{l \cdot n}$. Therefore we get:

$$\sum_{S \in OPT \cap V_0} gain(S) \;\leq\; l \cdot n \cdot \frac{gain_{max}}{l \cdot n} \;\leq\; w(M^*) - w(M). \tag{2}$$

We are now able to prove the main statement about the weight of the matching M' that is returned by the algorithm `ImproveMatching`.

Theorem 1 *If the algorithm* `ImproveMatching` *gets a matching M as input then it returns a matching M' such that*

$$w(M') \;\geq\; w(M) + \frac{1}{4l} \cdot \left(\frac{l-1}{l} \cdot w(M^*) - w(M) \right) \;.$$

Proof. The algorithm `ImproveMatching` computes a maximal independent set in the graph $G' \setminus V_0$ which consists of maximal independent sets in the graphs G'_i. Therefore it must be the case that for every augmentation S in $OPT \setminus V_0$ there exists an augmentation A in ALG with $gain(S) \leq 2 \cdot gain(A)$ and A and S are connected by an edge in G'. This means that the augmentations A and S must have a vertex v in G in common and we assign S to one such vertex v. By definition of the set OPT each vertex of G is contained in at most l augmentations and therefore at most l augmentations can be assigned to it. Each augmentation of ALG has at most $2l$ vertices. Therefore it follows that

$$gain(ALG) \;\geq\; \frac{1}{2}\frac{1}{2l^2} \sum_{S \in OPT \setminus V_0} gain(S) \;. \tag{3}$$

For the sum appearing at the right hand side of (3) we get the following inequality by using (1) and (2):

$$\begin{aligned} \sum_{S \in OPT \setminus V_0} gain(S) &= \sum_{S \in OPT} gain(S) - \sum_{S \in OPT \cap V_0} gain(S) \\ &\geq l \cdot w(M^*) - (l+1) \cdot w(M) - (w(M^*) - w(M)) \\ &= (l-1) \cdot w(M^*) - l \cdot w(M) \qquad (4) \end{aligned}$$

This paper appeared in: Information Processing Letters, 99:3 (2006), 119-123

Now combining (3) and (4) we get

$$\begin{aligned} gain(ALG) &\geq \frac{1}{2} \cdot \frac{1}{2l^2} \cdot l \cdot \left(\frac{l-1}{l} \cdot w(M^*) - w(M) \right) \\ &= \frac{1}{4l} \cdot \left(\frac{l-1}{l} \cdot w(M^*) - w(M) \right) \end{aligned}$$

As we have $w(M') = w(M) + gain(ALG)$ this proves the result. □

The following result now shows that we get an NC algorithm that finds a matching of weight at least $(1-\epsilon) \cdot w(M^*)$ by making a constant number of calls to the algorithm `ImproveMatching`.

Theorem 2 *For every $\epsilon > 0$ there exists an NC algorithm that finds in a weighted graph a matching of weight at least $(1-\epsilon) \cdot w(M^*)$.*

Proof. Let M_0 be the empty matching and M_{i+1} be the matching that is obtained from M_i by applying the algorithm `ImproveMatching`. By Theorem 1 we have $w(M_{i+1}) \geq w_{i+1} \cdot w(M^*)$ where w_{i+1} is defined by the following recurrence

$$w_{i+1} = w_i + \frac{1}{4l} \cdot \left(\frac{l-1}{l} - w_i \right), \text{ and } \quad w_0 = 0 .$$

By solving this linear recurrence equation (see for example [11]) we get

$$w(M_i) \geq \frac{l-1}{l} \cdot \left(1 - \left(1 - \frac{1}{4l} \right)^i \right) \cdot w(M^*) .$$

Now by setting for example $l = \frac{2}{\epsilon}$ we get

$$w(M_i) \geq \left(1 - \frac{\epsilon}{2} \right) \cdot \left(1 - \left(1 - \frac{\epsilon}{8} \right)^i \right) \cdot w(M^*) .$$

which immediately shows that if i is larger than some constant c (which is bounded by $O(1/\epsilon)$) we have

$$w(M_c) \geq (1-\epsilon) \cdot w(M^*) .$$

The algorithm `ImproveMatching` is called a constant number of times. Within one call the ranked augmentation graph G' can be calculated in $O(\log n)$ time using $O(n^{4l})$ processors as shown in Section 3. The while-loop is executed $O(\log n)$ times. In each iteration a maximal independent set needs to be computed in a graph with at most n^{2l} vertices. The algorithm of Goldberg and Spencer [4] allows to compute such an independent set in $O(\log^4 n)$ time using $O(n^{4l})$ processors. Lines 5 and 6 of the algorithm `ImproveMatching` can obviously be executed in $O(\log n)$ time using $O(n^{2l})$ processors. In total our algorithm needs $O(n^{8/\epsilon})$ processors using $O(\frac{1}{\epsilon}\log^5 n)$ time.

$\square$

6 Conclusion

Our algorithm needs $n^{O(\frac{1}{\epsilon})}$ processors. This is the same amount of processors that is needed in the $1-\epsilon$ NC-approximation algorithm for the unweighted case [3]. The dependence on ϵ can be improved by defining the ranking of the vertices suitably depending on ϵ. However, this makes the analysis much more complicated so we did not do it in this paper.

References

[1] D.E. Drake, S. Hougardy, Improved linear time approximation algorithms for weighted matchings, In: Approximation, Randomization, and Combinatorial Optimization, (Approx/Random) 2003, S.Arora, K.Jansen, J.D.P.Rolim, A.Sahai (Eds.), LNCS 2764, Springer 2003, 14–23.

[2] D.E. Drake, S. Hougardy, A linear time approximation algorithm for weighted matchings in graphs, ACM Transactions on Algorithms 1 (2005).

[3] T. Fischer, A.V. Goldberg, D.J. Haglin, S. Plotkin, Approximating matchings in parallel, Information Processing Letters 46 (1993), 115–118.

[4] M. Goldberg, T. Spencer, A new parallel algorithm for the maximal independent set problem, SIAM Journal on Computing 18:2 (1989), 419–427.

[5] J. Jájá, An Introduction to Parallel Algorithms, Addison-Wesley, Reading, Massachusetts 1992.

[6] H.J. Karloff, A Las Vegas RNC algorithm for maximum matching, Combinatorica 6:4 (1986), 387–391.

[7] M. Karpinski, W. Rytter, Fast Parallel Algorithms for Graph Matching Problems, Clarendon Press, Oxford 1998.

[8] R.M. Karp, E. Upfal, A. Wigderson, Constructing a Perfect Matching is in Random NC, Combinatorica 6:1 (1986), 35–48.

[9] S. Micali and V.V. Vazirani, An $O(\sqrt{V}E)$ Algorithm for Finding Maximum Matching in General Graphs, Proc. 21st Annual IEEE Symposium on Foundations of Computer Science (1980) 17–27.

[10] K. Mulmuley, U.V. Vazirani, V.V. Vazirani, Matching is as easy as matrix inversion, Combinatorica 7:1 (1987), 105–113.

[11] P.W. Purdom, C.A. Brown, The Analysis of Algorithms, Holt, Rinehart and Winston, New York, 1985.

[12] R. Uehara, Z.-Z. Chen, Parallel approximation algorithms for maximum weighted matching in general graphs, Information Processing Letters 76:1-2 (2000), 13–17.

[13] V.V. Vazirani, A Theory of Alternating Paths and Blossoms for Proving Correctness of the $O(\sqrt{V}E)$ General Graph Maximum Matching Algorithm, Combinatorica 14:1 (1994), 71–109.