Parallel Algorithms for the Assignment and Minimum-Cost Flow Problems

James B. Orlin *
Sloan School of Management
M.I.T.
Cambridge, MA

Clifford Stein †
Department of Math and Computer Science
Dartmouth College

Abstract

Let $G = (V, E)$ be a network for an assignment problem with $2n$ nodes and m edges, in which the largest edge cost is C. Recently the class of instances of bipartite matching problems has been shown to be in RNC provided that C is $O(\log^k n)$ for some fixed k. We show how to use scaling so as to develop an improved parallel algorithm and show that bipartite matching problems are in the class RNC provided that $C = O(n^{\log^k n})$ for some fixed k. We then generalize these results to minimum-cost flow problems. Let U be an upper bound on the capacities of the edges and on the largest demand. We show that the minimum-cost flow problem is in the class RNC, provided that $\log(C + U) = O(\log^k n)$ for some fixed k. Thus the minimum-cost flow problem is in the class RNC even when the magnitude of the costs and capacities are allowed to grow faster than any polynomial in n. The key to our approach is to reduce the number of processors needed from an amount that is proportional to the *magnitude* of the largest edge cost to an amount that is *independent* of the magnitude of the largest edge cost. The tradeoff is an increase in the running time that grows linearly in $\log(C + U)$.

Keywords: matching, assignment, parallel algorithm, scaling, minimum-cost flow

*Research partially supported by NSF Grant DDM-8921835 and by Grant AFOSR-88-0088 from the Air Force Office of Scientific Research.

†Support provided by NSF PYI Award CCR-89-96272 with matching support from UPS and Sun and by an AT&T Bell Laboratories Graduate Fellowship. This work was done while this author was at MIT Laboratory for Computer Science.

1 Introduction

Karp, Upfal, and Wigderson [12], and Mulmuley, Vazirani, and Vazirani [15], have recently developed randomized NC (RNC)[1] parallel algorithms for the minimum-cost perfect matching problem, assuming that the input is given in unary. For both of these algorithms, the number of processors needed is proportional to the *magnitude* of the largest number. In this note, we show how to convert these algorithms into algorithms that use a number of processors that is *independent of the magnitude of the largest number*, provided that the graph is bipartite. As a tradeoff, we get an increase in the time spent that is proportional to the logarithm of the magnitude of the largest edge cost. Let n be the number of nodes in the graph, and C the largest cost in the matching problem. If $C = \Omega(n^{1+\epsilon})$, we get algorithms that do less work, where work is the product of the number of processors used and the time spent. Assuming that $C = O(n^{\log^k n})$ for some constant k, our algorithms are in RNC. We achieve these results by using *scaling*, which reduces the problem of finding a matching in a graph with large edge costs to the problem of finding a sequence of matchings in a sequence of graphs, each of which has small edge costs. The algorithm is similar to the sequential algorithms of [6] and [8], as it iteratively finds an assignment and dual variables and then uses these to ensure that during the next iteration the graph contains a matching of small total cost. By iterating this algorithm on an appropriately derived graph, we also obtain improved results for the minimum-cost flow problem.

Our model of computation is a parallel random-access machine (PRAM) [4]. We define the *work* done by a parallel algorithm as the product of the number of processors used and the time spent.

2 Preliminaries

Let $G = (V, E, c)$ be an undirected bipartite graph with node set V, edge set E, and an integral *cost* $c(v, w)$ associated with each edge (v, w). We will denote the two sets of nodes in the bipartition as V_1 and V_2, and let $n = |V_1| = |V_2|$ and $m = |E|$.

A *matching* on a graph is a set M of edges, such that each node is incident to no more than one edge from M. A *perfect matching* is a matching in which every node is incident to exactly one matched edge. If the edges have costs, the cost of a matching is the sum of the costs of the edges in the matching. A *minimum-cost perfect matching* (MCPM) is the perfect matching with the smallest possible cost. In a bipartite graph, an MCPM is also

[1] NC is the class of algorithms that, on input of size n, use n^{k_1} processors and $O(\log^{k_2} n)$ time, for some constants k_1 and k_2. RNC algorithms are NC algorithms that allow each processor to generate an $O(\log n)$ bit random number at each step in the computation and return the correct answer with probability greater than 1/2.

called an *assignment*. It will be convenient to associate an integer-valued dual variable $d(v)$ with each node v. This allows us to define $c_d(v,w)$, the *reduced cost of edge* (v,w), *with respect to dual variables* d by $c_d(v,w) = c(v,w) + d(v) - d(w)$. Let M be a matching. We say that a set of dual variables is *tight* if

$$c_d(v,w) \geq 0 \quad \forall (v,w) \in E \tag{1}$$

$$c_d(v,w) = 0 \quad \forall (v,w) \in M \tag{2}$$

The first algorithm for the assignment problem was Kuhn's *Hungarian algorithm* [13]. Implemented with Fibonacci heaps [5], this algorithm runs in $O(nm + n^2 \log n)$ time, which remains the fastest strongly polynomial algorithm for the assignment problem. Gabow and Tarjan [8] have developed an algorithm that runs in $O(\sqrt{n}m \log(nC))$ time. There are no known NC algorithms for the assignment problem; however, there are RNC algorithms under the assumption that the input is given in unary. The first RNC algorithm under this assumption was given by Karp, Upfal, and Wigderson [12]. An implementation of this algorithm by Galil and Pan [9] uses $(n + C')M(n)$ processors and $O(\log n \log^2(nC'))$ time where C' is an upper bound on the maximum cost of any matching, and $M(n)$ is the minimum number of processors needed to multiply two $n \times n$ matrices. Currently $M(n) = O(n^{2.376})$ [2], and trivially, $M(n) = \Omega(n^2)$. Subsequently, a faster algorithm was discovered by Mulmuley, Vazirani, and Vazirani [15], that finds an assignment in $O(\log^2 n)$ time using $nmCM(n)$ processors, where C is the largest edge cost in the input graph. As neither one of these algorithms does less work than the other on all graphs, we will give our improvements relative to both of these algorithms. Other parallel algorithms for the assignment problem include [7], [10] and [17].

3 A Scaling Algorithm

Our algorithm is a scaling algorithm, similar in general structure to the sequential matching algorithm of Gabow and Tarjan [8]. The algorithm proceeds in $\log C$ iterations. At the beginning of each iteration, one bit is added to the costs and dual variables. Then a perfect matching and tight dual variables are found on the graph with edges of reduced cost no greater than $2n$. The new dual variables are added to the old ones and the iteration terminates. The details of the algorithm appear in Figure 1.

The algorithm of [8] works in a similar manner, except that in their algorithm each iteration involves finding only an approximate matching. At each step, we find an exact matching and tight dual variables. By working with reduced costs at each iteration, we ensure that not only is the cost of the new matching close to that of the old matching, but also that the magnitude of the edge costs used in each iteration is small. Thus we can

Input: $G = (V, E, \hat{c})$ an undirected bipartite graph with bipartition V_1 and V_2 and cost $\hat{c}(v, w)$ on edge (v, w). Assume that G contains a perfect matching.
Output: A minimum-cost perfect matching M.

1 $\quad d(v) \leftarrow 0 \; \forall v \in V_2$.
$\quad c(v, w) \leftarrow 0 \;\; \forall (v, w) \in E$.
$\quad$ Let $C = \max_{(v,w)\in E}\{|\hat{c}(v, w)|\}$.
2 $\quad$ For $l = 1$ to $\lceil \log_2 C \rceil$
3 $\quad\quad d(v) \leftarrow 2d(v) \forall v \in V$.
$\quad\quad c(v, w) \leftarrow 2c(v, w) +$ (the l^{th} signed bit of $\hat{c}(v, w)$) $\forall (v, w) \in E$.
4 $\quad\quad$ Let $E' = \{(v, w) \mid (v, w) \in E \text{ and } c_d(v, w) \leq 2n\}$.
5 $\quad\quad$ Compute M, a MCPM in $G' = (V, E', c_d)$.
6 $\quad\quad$ Compute tight dual variables for G' with respect to matching M using costs c_d. Let $\Delta(v)$ be the dual variable associated with v.
7 $\quad\quad d(v) \leftarrow d(v) + \Delta(v) \;\; \forall v \in V$.
8 $\quad$ Output M, a minimum-cost perfect matching.

Figure 1: Algorithm Assignment.

reduce the assignment problem to a series of $\log C$ assignment problems, each in a graph with small edge costs.

Before proceeding to analyze our algorithm, we address the issue of finding tight duals. In general, given an optimal flow for a minimum-cost flow problem, a single shortest-path computation suffices to find tight dual variables (see, for example [1]). As specialized to the assignment problem the algorithm is as follows. First create a directed residual network $G' = (V', E')$ such that

- for each $(v, w) \in E$, there is an edge $(v, w) \in E'$ with cost $c(v, w)$,
- for each $(v, w) \in M$, there is an edge $(w, v) \in E'$ with cost $-c(v, w)$,
- a new node $s \in V_1$, and for each node $w \in V_2$, an edge (s, w) of cost nC.

Now let $d(v)$ be the shortest path distance from s to all other nodes. It is well known that the $d(v)$ that result are tight dual variables for the matching problem.

We now demonstrate that Algorithm Assignment described in Figure 1 is correct and efficient. The key to the efficiency of this algorithm is that when we find a matching in step 4, we may ignore edges with reduced cost greater than $2n$. It remains to be shown that this does not change the value of the MCPM.

Lemma 3.1 *The graph $G = (V, E', c_d)$ formed in step 4 of algorithm* Assignment *always contains a MCPM of total cost no more than n. Further, $\forall (v, w) \in E$, $c_d(v, w) \geq -1$.*

Proof: We will prove this by induction on the number of executions of step 4. During the first execution, all reduced edge costs are either -1, 0 or 1, so the lemma is true. Assume that it is true after step 4 on iteration $l-1$. Then, at the beginning of the loop, the costs c_d are tight with respect to Δ. Because for all edges (v,w)

$$c_d(v,w) + \Delta(v) - \Delta(w) = (c(v,w) + (d(v) + \Delta(v)) - (d(w) + \Delta(w))$$

it is also true that c is tight with respect to $d + \Delta$.

Thus, in the graph $\hat{G} = (V, E, c)$, the reduced cost of the edges of M with respect to $d+\Delta$ is 0 and the reduced cost of every edge is non-negative. Let $d \leftarrow d+\Delta$. Now consider the effect of adding a new bit of cost and updating the dual variables in Step 3 of iteration l. After multiplying all dual variables by 2 and all costs by 2, the reduced cost of each edge is still non-negative and the reduced cost of each edge in M is still 0. After adding a new-bit of cost, each edge has reduced cost greater than or equal to -1, and each edge in M has reduced cost at most 1. Therefore, the sum of the cost of the edges in M is at most n. ■

Corollary 3.2 *In the graph $G = (V, E', c_d)$ formed in step 4 of algorithm* Assignment*, there always exists a MCPM M in which $c_d(v,w) \leq 2n\ \forall (v,w) \in M$.*

Proof: Assume, to the contrary, that the MCPM contains an edge of value greater than $2n$. By Lemma 3.1 every other edge in the matching has at least -1. Therefore this matching has value greater than $2n + (n-1)(-1) > n+1$. But by Lemma 3.1, we know that there exists a perfect matching of value at most n, therefore the one we have cannot be the minimum one. ■

This leads to our main result.

Theorem 3.3 *Let algorithm A be a randomized parallel algorithm for MCPM that uses $Cf(n,m)$ processors and $O(\log^k n)$ time, where $f(n,m)$ is a polynomial in n and m and k is a non-negative integer. Using algorithm* Assignment *we can convert algorithm A into an algorithm for MCPM that uses $nf(n,m) + M(n)$ processors and $O((\log^k n + \log^2 n)\log C)$ time.*

Proof: First we must verify that our algorithm actually finds a MCPM. From Corollary 3.2, we see that ignoring edges of reduced cost greater than $2n$ does not change the value of the MCPM. Therefore, at each step we find a valid MCPM with respect to the reduced costs. Because an MCPM with respect to the reduced costs has the same value as an MCPM with respect to the actual costs, in the last iteration we really are finding an MCPM in the graph where the current edge costs are the same as the edge costs of the input graph, thus proving correctness. To derive the resource bounds, observe that whenever we find a MCPM in step 5, $C \leq 2n$. Procedure *Compute Tight Duals* is dominated by the shortest path computation that takes $O(\log^2 n)$ time on $M(n)$ processors. All other steps in the algorithm can be implemented in constant time on $O(m+n)$ processors. Combining these observations with the fact that there are only $\log C$ iterations of the main loop, the theorem follows. ■

Corollary 3.4

- *Algorithm* Assignment*, combined with the matching algorithm of [12], yields a randomized parallel algorithm for computing an MCPM using* $n^2 M(n)$ *processors and* $O(\log^3 n \log C)$ *time.*
- *Algorithm* Assignment*, combined with the matching algorithm of [15], yields a randomized parallel algorithm for computing an MCPM using* $n^2 m M(n)$ *processors in* $O(\log^2 n \log C)$ *time.*

Proof: Immediate from Theorem 3.3 and the algorithms in [12], [9], and [15]. ■

Observe that our algorithm performs less work in the case that $C = \Omega(n^{1+\epsilon})$ for some $\epsilon > 0$. Further, our algorithm outperforms the old algorithms by a factor of $O(\frac{C}{n \log C})$, so as C gets larger, our algorithm becomes even more efficient than the previous algorithms.

We can extend this algorithm for a minimum-cost perfect matching to one that finds a minimum-cost (not necessarily perfect) matching. (We allow the edge costs to be negative. Otherwise, the optimum is the null matching.) Let $G = (V, E, c)$ be a graph in which we would like to find a minimum-cost matching. We employ the standard transformation of using an augmented graph $G' = (V, V_1 \times V_2, c')$ where $c'(v, w) = c(v, w)$ if $(v, w) \in E$ and $c'(v, w) = 0$ otherwise. It is easy to see that if M' is a minimum-cost perfect matching in G', then $M' \cup E$ is a to a minimum-cost matching in G.

Corollary 3.5 *Given a graph G with maximum edge cost C, running algorithm* Assignment *on G' yields an algorithm that finds a minimum-cost matching using $n^2 M(n)$ processors and $O(\log^3 n \log C)$ time or $n^4 M(n)$ processors and $O(\log^2 n \log C)$ time.*

4 Minimum Cost Flow Problems

Our technique also yields improved results for the *minimum-cost flow problem.* We assume familiarity with the minimum-cost flow problem as we will only sketch the ideas needed to extend the class of minimum-cost flow instances that are in RNC. We refer the reader to [1, 14, 11] for more detail on minimum-cost flows.

The first idea is to show that a general minimum-cost flow problem can be reduced to a series of minimum-cost flow problems, each with small edge capacities. This fact is implicit in the algorithm of Edmonds and Karp [3], here we make it explicit.

Lemma 4.1 *Let P be a minimum-cost flow problem with n nodes, m edges, maximum edge cost C and maximum edge capacity U. Then P can be, in NC, reduced to the solution of $O(\log U)$ minimum-cost circulation problems, each with n nodes, m edges, maximum edge cost C and maximum edge capacity m, and total supply m.*

Proof: As in [3], we will introduce the capacities one bit at a time. We will now describe how to convert a minimum-cost flow with respect to the first ℓ bits of the capacities to one with respect to the first $\ell + 1$ bits. Given a flow f that is minimum-cost with respect to capacities u, we first double the flow and double the capacities. This still gives a minimum-cost flow. Next we introduce a new bit of capacity. The new flow may no longer be minimum-cost, but as is discussed in [3] it is "close" to a minimum-cost flow. We now bring the edges "in-kilter" by saturating residual edges with negative reduced-cost. This introduces supplies and demands at nodes, but since we have added at most one unit of capacity to each edge, the sum of all the supplies is most m. Further, since no edge capacity need be more than the sum of supplies, we can limit all edge capacities to m also. So we now have a minimum-cost circulation problem with n nodes, m edges, maximum edge cost C, maximum edge capacity m, and total supply m. We discuss below how to solve such a problem. Since the initial capacities are at most U, we need repeat this process at most $\lceil \log U \rceil$ times. ■

We are now left with the problem of solving a minimum-cost circulation problem with n nodes, m edges, maximum edge cost C and maximum edge capacity m, and total supply m. Using a standard transformation, (see, for example [1], [16]), we can convert this capacitated flow problem into an uncapacitated transportation problem with $m + n$ nodes, $2m$ edges, maximum demand m and maximum cost C. We now use another standard transformation (see, for example [14]) that converts an uncapacitated transportation problem into an assignment problem. If node v has demand d, we make $d(v)$ copies of that node. For each edge (v, w) we put an edge between every copy of v and every copy of w. This creates an assignment problem with $O(m^2)$ nodes, $O(m^4)$ edges, and maximum edge cost C. We now apply algorithm ASSIGNMENT to solve this problem.

Theorem 4.2 *We can solve a minimum-cost flow problem using $m^4 M(m^2)$ processors and $O(\log^3 n \log C \log U)$ time or using $m^5 M(m^2)$ processors and $O(\log^2 n \log C \log U)$ time.*

When $C = \Omega(m^2)$, this compares favorably with the previous results of $Cm^2 M(m^2)$ processors and $O(\log^3 n \log U)$ time or $Cm^3 M(m^2)$ processors and $O(\log^2 n \log U)$ time.

Acknowledgments

We are grateful to David Shmoys and Joel Wein for helpful discussions, and to Margaret Tuttle and David Williamson for reading an earlier draft of this paper.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. Network flows. In G.L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, editors, *Handbook in operations research and management science, Volume 1: Optimization*, pages 211–360. North-Holland, Amsterdam, 1990.

[2] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progression. *J. Symbolic Computation*, 9:251–280, 1990.

[3] J. Edmonds and R.M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19:248–264, 1972.

[4] S. Fortune and J. Wyllie. Parallelism in random access machines. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*, pages 114–118, 1978.

[5] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34:596–615, 1987.

[6] H. Gabow. Scaling algorithms for network problems. *Journal of Computer and System Sciences*, 31:148–168, 1985.

[7] H. N. Gabow and R. E. Tarjan. Almost-optimum parallel speed-ups of algorithms for bipartite matching and related problems. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 514–527, 1988. To appear in *Journal of Algorithms*.

[8] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18:1013–1036, 1989.

[9] Z. Galil and V. Pan. Improved processor bounds for algebraic and combinatorial problems in $\mathcal{RNC}$. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, pages 490–495, 1985. To appear in *Journal of the ACM*.

[10] A. V. Goldberg, S. A. Plotkin, and P. M. Vaidya. Sublinear-time parallel algorithms for matching and related problems. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 174–185, 1988.

[11] A. V. Goldberg, R. E. Tarjan, and /'E. Tardos. Network flow algorithms. Technical Reprot STAN-CS-89-1252, Department of Computer Science, Stanford University, Stanford, CA, March 1989.

[12] R. Karp, E. Upfal, and A. Wigderson. Constructing a perfect matching is in random $\mathcal{NC}$. *Combinatorica*, 6:35–48, 1986.

[13] H.W. Kuhn. The hungarian method for the assignment problem. In *Naval Research Logistics Quarterly*, volume 2, pages 83–97, 1955.

[14] E.L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, 1976.

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

[16] J.B. Orlin. A faster strongly polynomial minimum cost flow problem. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 377–387, 1988.

[17] C.N.K. Osiakwan and S. Akl. A perfect speed-up parallel algorithm for the assignment problem on complete weighted bipartite graphs. In *Proceeding of IEEE Parbase 90*, pages 293–304, 1990.