Expected Time Complexity of the Auction Algorithm and the Push Relabel Algorithm for Maximum Bipartite Matching on Random Graphs*

Oshri Naparstek, Amir Leshem
School of Engineering, Bar-Ilan University, 52900, Ramat-Gan, Israel

Received 2 August 2013; revised 26 March 2014; accepted 21 August 2014
Published online 23 December 2014 in Wiley Online Library (wileyonlinelibrary.com).
DOI 10.1002/rsa.20578

ABSTRACT: In this paper we analyze the expected time complexity of the auction algorithm for the matching problem on random bipartite graphs. We first prove that if for every non-maximum matching on graph G there exist an augmenting path with a length of at most $2l + 1$ then the auction algorithm converges after $N \cdot l$ iterations at most. Then, we prove that the expected time complexity of the auction algorithm for bipartite matching on random graphs with edge probability $p = \frac{c \log(N)}{N}$ and $c > 1$ is $O\left(\frac{N \log^2(N)}{\log(Np)}\right)$ w.h.p. This time complexity is equal to other augmenting path algorithms such as the HK algorithm. Furthermore, we show that the algorithm can be implemented on parallel machines with $O(\log(N))$ processors and shared memory with an expected time complexity of $O(N \log(N))$.
 Random Struct. Alg., 48, 384–395, 2016

Keywords: complexity; auction algorithm, Pushrelabel algorithm; Bipartite matching; random graphs

1. INTRODUCTION

One of the most extensively studied problems in combinatorial optimization in the last 50 years known as the Bipartite Maximum Cardinality Matching (BMCM). The main goal of the BMCM problem is to find an assignment where the maximum number of vertices are matched on bipartite graphs, thus making it a special case of the max-flow problem as well as the max-sum assignment problem. Hence, any algorithm that solves one of these problems also solves the BMCM problem. Most of the algorithms that solve the BMCM

Correspondence to: Oshri Naparstek
*This research was funded by ISF grant 903/2013.

problem are augmenting paths algorithms.The Hopcroft-Karp (HK) algorithm [15] is one of the most well-studied of these. The HK algorithm has a worst case time complexity of $O\left(\sqrt{N}m\right)$ where N is the number of vertices in the bipartie graph and m is the number of edges in the graph. The HK algorithm was later revisited by Feder and Motwani [11] and was proven to have a worst case time complexity of $O\left(\frac{\sqrt{N}m}{\log(N)}\right)$. Recently, an algorithm with an improved complexity of $O\left(N^{\frac{10}{7}}\right)$ was introduced by Madry [17]. Another well known augmenting path algorithm that solves the BMCM problem is Dinic's algorithm [9]. Motwani [18] proved that Dinic's algorithm achieves perfect matching on random bipartite graphs with an expected running time of $O\left(\frac{m\log(N)}{\log\Delta}\right)$ where Δ is the expected number of neighbours per vertex. It was later shown by Bast et al. [2] that algorithms that use the shortest augmenting paths, have an expected time complexity of $O\left(\frac{\sqrt{N}m}{\log(N)}\right)$. Recently it was shown by Chebolu et al. [7] that maximum cardinality matching can be found on sparse random graphs with an expected time complexity of $O(m)$ using a combination of the Karp-Sipser heuristic and an augmenting path finding heuristic. Goel et al. [12] presented an algorithm with an expected time of $N\log(N)$ iterations on regular bipartite graphs.

Other solutions to the BMCM problem that do not use the augmenting path method include the auction algorithm [5] and the push relabel (PR) algorithm [14]. The main difference between the augmenting path algorithms described above and these algorithms is that in augmenting path algorithms, an augmenting path is first found and then augmented. By contrast, in the auction and PR algorithms, only two edges are augmented on each iteration according to some update rule. It was shown by both Goldberg [13] and Bertsekas [6] that the push relabel algorithm and the auction algorithm are equivalent. Since these algorithms are equivalent we will refer only to the auction algorithm from now on. The average time complexity of the auction algorithm is not known and the problem of calculating it has remained unresolved more than 30 years. However, in many cases it has been shown that the auction algorithm converges faster than other methods in solving the assignment problem [4]. Furthermore, several experimental studies [8, 16, 19, 20] showed that in practice, auction algorithms outperforms augmenting paths based algorithms on many real life scenarios for the solution of the BMCM problem. Bertsekas, Castanon, and Tsaknakis, Reverse auction and the solution of inequality constrained assignment problems, Unpublished Report, 1991, conjectured that the average running time of the auction algorithm would be $O(m\log(N))$ on bipartite graphs with uniformly distributed weights where m is the number of edges in the bipartite graph and N is the number of vertices on each side of the graph.

In this paper we analyze the average time complexity of the auction algorithm for the BMCM problem. We first prove that if for every non-maximum matching on graph G there exist an augmenting path with a length of at most $2l+1$ then the auction algorithm converges after $N\cdot l$ iterations at most.

Then, we prove that the expected time complexity of the auction algorithm for random bipartite graphs where each edge is independently selected with probability $p=\frac{c\log(N)}{N}$ and $c>2$ is $O\left(\frac{N\log^2(N)}{\log(Np)}\right)$. We show that by reducing the density of the graph such that $p=\frac{c\log N}{N}$ the complexity simplifies to $O\left(\frac{N\log^2(N)}{\log(\log(N))}\right)$. We then present a parallel implementation of the algorithm for parallel machines with $O(\log(N))$ processors and a shared memory. We prove that the expected time complexity of the parallel implementation is $O(N\log(N))$.

2. THE AUCTION ALGORITHM

The auction algorithm [5] is an intuitive method for solving the assignment problem. Auctions in which unassigned people raise their prices and bid for objects simultaneously was the original inspiration for the auction algorithm. Similarly, the auction algorithm has two stages, the bidding stage and the assignment stage. In the bidding stage each unassigned individual raises the price of the object he wishes to acquire by the difference between the most profitable object and the second most profitable object plus some constant ϵ. In the assignment stage every object is assigned to the highest bidder. The two stages are repeated until all bidders are assigned an object. More specifically, let $\mathbf{R}$ be the matrix of the initial rewards. Let $\mathbf{B}$ be a matrix of the bids. $\boldsymbol{\rho}$ is the price vector where ρ_k is given by:

$$\rho_k = \max_n \mathbf{B}(n,k) \tag{1}$$

Let $\boldsymbol{\eta} = [\eta_1, \eta_2, \ldots, \eta_N]$ be an assignment (permutation) vector where η_n is the object that is assigned to the n'th person; i.e., the matching $\{(U_n, V_{\eta_n}) : n = 1..N\}$ is a perfect matching for the complete bipartite graph $K_{n,n}$.

$$(U_n, V_{\eta_n}) \in M$$

Definition 1. *An object k is said to be* assigned *to person n by* $\boldsymbol{\eta}$ *if* $\boldsymbol{\eta}_n = k$.

The reward of the n'th person on assignment $\boldsymbol{\eta}$ is denoted by $\mathbf{R}(n, \eta_n)$ and the price that the n'th person pays on assignment $\boldsymbol{\eta}$ is denoted by ρ_{η_n}. Given a positive scalar ϵ, an assignment η and a price $\rho_{\eta n}$, a person n is termed *happy* with assignment $\boldsymbol{\eta}$ if the profit (i.e., reward minus price) is within ϵ of the maximum profit achievable by person n. This condition is called ϵ-Complementary slackness (ϵ-CS).

$$\mathbf{R}(n, \eta_n) - \rho_{\eta n} \geq \max_k (\mathbf{R}(n,k) - \rho_k) - \epsilon \tag{2}$$

When all the people are assigned and happy, the algorithm stops. It was shown in [5] that the algorithm terminates in finite time. The algorithm is within $N\epsilon$ of being optimal at termination [5]. Also, if the initial prices are all zeros and $N \leq K$ the algorithm still converges in finite time and with the same bounds on optimality.

3. A SIMPLIFIED AUCTION ALGORITHM FOR MAXIMUM BIPARTITE MATCHING

The BMCM problem is a special case of the assignment problem where the values in the reward matrix are either 0 or 1. Furthermore, since the reward matrix is integer valued, the choice of $\epsilon = \frac{1}{N+1}$ is sufficient for obtaining an optimal solution [5]. The cardinality of a matching is the number of vertices that were assigned to edges with weight 1. Here we simplify the auction algorithm and assume that on each iteration an unassigned person is picked and raises the price of his most desirable object by exactly ϵ. Under this assumption, the price will be:

$$\rho_k = \frac{h_k}{N+1} \tag{3}$$

TABLE 1. Simplified auction algorithm for maximum matching in bipartite graphs

1. Initialize $h_v = 0, \forall v \in V$ and set $M = \emptyset$
2. While $|M| < N$ and $\sum_{k=1}^{N} h_k < N(N-1)$ do
   2.1 Choose a free vertex $u \in U$
   2.2 $j = \arg\min_{v \in n_u^G} h_v$
   2.3 Let u_{old} be the match of j under M
   2.4 $M = M \setminus (u_{old}, j)$
   2.5 $M = M \cup (u, j)$
   2.6 $h_j = h_j + 1$
3. return

where h_k is the number of times that the price of the k'th object was raised. We further simplify the algorithm and assume that a person would only bid on objects with positive rewards. Under this assumption the ϵ-CS condition (2) for an object with a positive reward becomes:

$$1 - \frac{h_{\eta_n}}{N+1} \geq \max_k \left\{ \left(1 - \frac{h_k}{N+1} \right) - \frac{1}{N+1} \right\}. \tag{4}$$

Multiplying both sides by $N+1$ and rearranging yields a simplified ϵ-CS condition for the BMCM problem

$$h_{\eta_n} - 1 \leq \min_k (h_k). \tag{5}$$

Using the simplified ϵ-CS condition we can derive a simplified version of the auction algorithm for the BMCM problem. On each iteration an unassigned person is chosen, the person is assigned to the k'th object with minimal h_k and raises it by 1. The description of the algorithm using graph theory terminology is as follows: Let $G = (U, V, E)$ be a bipartite graph with vertex sets U, V where $|U| = |V| = N$ and an edge set $|E|$. Let $\mathbf{h} = [h_1, h_2, \ldots, h_N]$ be the value assigned to the vertices in V and define n_v^G to be the set of neighbors of vertex $v \in U \cup V$ in the graph. On each iteration a free vertex $u \in U$ is assigned to a vertex $v \in V$ with minimal value h_k. If another vertex was previously assigned to that vertex it becomes free. The algorithm stops after a perfect matching is found or the number of iterations is larger than $N^2 - N$. The algorithm is depicted in Table 1. Note that the simplified auction algorithm is equivalent to the push relabel algorithm with double push [13].

4. THE EXPECTED NUMBER OF ITERATIONS OF THE SIMPLIFIED AUCTION ALGORITHM FOR THE BMCM PROBLEM

In this section we analyze the expected time complexity of the simplified auction algorithm for the BMCM problem. Unless otherwise mentioned, G is a graph as described at the end of section 3. We start by the following useful observation:

Observation 1. *Let T be the number of iterations until the algorithm terminates and let h_v be the value of vertex v at termination, then*

$$T = \sum_{v=1}^{N} h_v \tag{6}$$

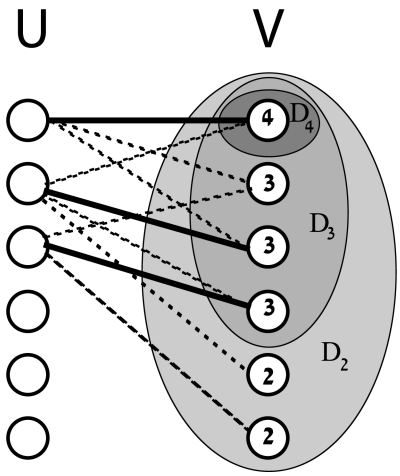

Fig. 1. An illustration of lemma 1. Here, the bold edges represent the edges in the matching. The top edge is a part of the matching and the top right vertex is in D_4. From lemma 1 it follows that all of the neighbours of the top left vertex are in D_3.

From the above observation it follows that T can be bounded if h_v is bounded for all $v \in V$. Hence, in the next lemmas we will analyze the development of the values h_v over the course of the simplified auction algorithm. We will prove that if the shortest augmenting path over all matching is equal or less than $2l+1$ then for all $v \in V$ $h_v \leq l$. We will then use the results of Motwani [18] to show that w.h.p this is the case for random bipartite graphs. In the proof we will use the following definitions:

Definition 2. *Let $h_v(i)$ be the value of vertex v in the i'th iteration of the algorithm.*

Definition 3. *Let $D_l(i)$ be a subset of V defined by $D_l(i) = \{v \in V : h_v(i) \geq l\}$.*

From the definition of $D_l(i)$ it follows that $D_l(i) \subseteq D_{l-1}(i) \subseteq ... \subseteq D_0(i)$.

Lemma 1. *Let $M(i) \subseteq E$ be a matching obtained by the algorithm in the i'th iteration. If $v_0 \in D_l(i)$ and $(u, v_0) \in M(i)$ then $n_u^G \subseteq D_{l-1}(i)$*

Proof. Let $v_0 \in D_l(i)$ and $(u, v_0) \in M(i)$. Then, there exist $0 \leq k < i$ such that $h_{v_0}(k) = l-1$ and $h_{v_0}(k+1) = l$. By the algorithm definition $v_0 = \arg\min_{v \in n_u^G} h_v(k)$. Hence, by the choice of v_0 it follows that $l-1 = h_{v_0}(k) \leq h_v(k)$ for all $v \in n_u^G$. As a result, this implies that in the k'th iteration $n_u^G \subseteq D_{l-1}(k)$. Since $h_v(n)$ is a non-decreasing function of n (prices never go down) it also follows that this is true for every $i > k$ because $n_u^G \subseteq D_{l-1}(k) \subseteq D_{l-1}(i)$. ∎

An illustration of lemma 1 is given in Fig. 1.

Claim 2. *Let $u_0 \in U$ be a vertex such that in the i'th iteration of the algorithm $n_{u_0}^G \subseteq D_l(i)$, $l \geq 2$. Let $u_1 \in U$ be the end point of an alternating path $P = (u_0, j, u_1)$ with $|P| = 2$ such that $(j, u_1) \in M$ then*

$$n_{u_1}^G \subseteq D_{l-1}(i) \tag{7}$$

Proof. We first observe that if $n^G_{u_0} \subseteq D_l(i)$, then by definition 3 for every vertex $v \in n^G_{u_0}$ it holds that $h_v(i) \geq l$ and therefore $v \in D_l(i)$. By the definition of an alternating path, if $n^G_{u_0} \subseteq D_l(i)$ there exists a vertex $j \in n^G_{u_1} \cap n^G_{u_0}$ such that $(u_1, j) \in M(i)$, $(u_0, j) \notin M(i)$ and $j \in D_l(i)$. Hence, by lemma 1 it implies that $n^G_{u_1} \subseteq D_{l-1}(i)$. ∎

Lemma 3. *Let $u_0 \in U$ be a free vertex such that in the i'th iteration of the algorithm $n^G_{u_0} \subseteq D_l(i)$; then every augmenting path in G for $M(i)$ starting from u_0 is at least of length $2l + 1$.*

Proof. For the case of $l = 0$ this is trivial. For the case of $l = 1$ this is also trivial since a vertex $v \in V$ is free only if $h_v(i) = 0$. Hence, if $n^G_{u_0} \subseteq D_l(i)$ there is no augmenting path of length 1 and the next shortest augmenting path is at least of length 3. For the case of $l \geq 2$ let $u_m \in U$ be the end point of an alternating path P with $|P| = 2m$ starting from u_0. We can apply claim 2 recursively and get that $n^G_{u_m} \subseteq D_{l-m}(i)$ and since $D_{l-m}(i) \subseteq D_{l-m-1}(i)$ then for every vertex $v \in n^G_{u_m}$

$$v \notin D_{l-m-1}(i) \setminus D_{l-m}(i). \tag{8}$$

The last vertex in an augmenting path is a free vertex. Note that a vertex $v \in V$ is free only if $v \in D_0(i) \setminus D_1(i)$. However, from equation (8) if $n^G_{u_0} \subseteq D_l(i)$ then for all $v \in n^G_{u_m}$ and $m \leq l - 2$ it follows that $v \notin D_0(i) \setminus D_1(i)$. Hence, if $m \leq l - 2$ there is no augmenting path of length $2m + 1$ starting from u_0. We can now conclude that the shortest augmenting path that starts from u_0 is at least $2m + 1$ with $m \geq l - 1$. However, when $m = l - 1$ then $u_m = u_{l-1}$ and $n^G_{u_{l-1}} \subseteq D_1(i)$. Using the case of $l = 1$ we get that the shortest augmenting path that starts from u_0 is at least of length $2(l - 1) + 3 = 2l + 1$ ∎

A well-known theorem by Berge [3] states that if a bipartite graph G contains a perfect matching there exists an augmenting path in G with respect to any non maximum matching M.

Theorem 4 ([3])**.** *If G contains a perfect matching there exists an augmenting path in G for any partial matching M.*

We will now use the above theorem and lemma 3 to prove the following lemma:

Lemma 5. *If G contains a perfect matching and for any non-perfect matching $M \subseteq E$ there exists an augmenting path of length at most $2l + 1$, then for each free vertex $u \in U$ and a partial matching $M(i)$ obtained by the algorithm in the ith iteration there exists at least one neighbor $j \in n^G_u$ such that $h_j(i) \leq l$ on each iteration of the algorithm.*

Proof. We will prove the lemma by contradiction. We will show that if at the i'th iteration of the algorithm there exist a vertex u with respect to $M(i)$ such that $j \in N_u$ and $h_j(i) > l$, then we can find a non-maximum matching that does not contain an augmenting path of length at most $2l + 1$. Assume towards contradiction that G contains a perfect matching and for any non-perfect matching $M \subseteq E$ there exists an augmenting path of length at most $2l + 1$ but on the i'th iteration there exists a free vertex u with respect to $M(i)$ such that if $j \in N_u$ then $h_j(i) > l$. This means by Lemma 1 that $n^G_u \subseteq D_{l+1}(i)$. By lemma 3 each augmenting path that begins at u is of length of at least $2l + 3$. However, our assumption is that for any non maximum matching M of G there exists at least one augmenting path of length $2l + 1$ or less. This implies that u is not the starting vertex of any augmenting

path with length of at most $2l+1$ in G for $M(i)$. This also implies that u is not the starting vertex of any augmenting path with length of at most $2l+1$ in G for $M(j)$ for any $j \geq i$. This is true since we know from lemma 1 that $D_m(i) \subseteq D_m(j), \forall m, \forall j \geq i$. Hence, following our assumptions there must always exist a vertex $\tilde{u} \neq u$ which is the starting vertex of an augmenting path with length of at most $2l+1$ in G for $M(j)$. Without loss of generality we can always choose $\tilde{u}$ on each iteration $j \geq i$. We can do this up to the point where $|M(j)| = N-1$ and the only free vertex is u. This implies that there is no augmenting path of length at most $2l+1$ to $M(j)$. This can occur in two situations:

1. No augmenting path exists to G for $M(j)$. Theorem 4 implies that no perfect matching exists for G in contradiction to the assumption that a perfect matching exists.
2. There exists an augmenting to G for $M(j)$ but with length at least $2l+3$. This contradicts our assumption that every non maximum matching contains an augmenting path of length at most $2l+1$

▪

Lemma 6. *If G contains a perfect matching and for any non-perfect matching $M \subseteq E$ there exists an augmenting path of length at most $2l+1$, then for every $v \in V$ at each iteration of the algorithm until the algorithm terminates*

$$h_v(i) \leq l+1, \forall i, v \in V \tag{9}$$

Proof. On each iteration i, the free vertex from U selects a vertex j such that $j = \arg\min_{v \in n_u^G} h_v(i)$ and set $h_j(i+1) = h_j(i) + 1$. However, we assume that G contains a perfect matching and for any non-perfect matching $M \subseteq E$ there exists an augmenting path of length at most $2l+1$. Hence, by lemma 5 to any free vertex at the i'th iteration with respect to a partial matching $M(i)$ there exists a vertex v_0 such that $h_{v_0}(i) \leq l$. This means $\min_{v \in n_u^G} h_v(i) \leq l, \forall i$. As a result, if there exists vertex $v_0 \in V$ where $h_{v_0}(i) = l+1$ then $\arg\min_{v \in n_u^G} h_v(i) \neq v_0$. This implies that if $h_{v_0}(i) = l+1$ its value never rises and as a result $h_v(i) \leq l+1, \forall i, v \in V$. ▪

Lemma 7. *If G contains a perfect matching then $T \leq N(N-1)$.*

Proof. G contains $2N$ vertices. Hence, every path is of length at most $2N-1$ and in particular every augmenting path is of length at most $2N-1$ for any non maximum matching. By lemma 6 we get that $h_v(i) \leq N-1, \forall i, v \in V$. Using lemma 1 we get that $T = \sum_{v=1}^{N} h_v \leq N(N-1)$. ▪

After the deterministic analysis of the algorithm, we now turn to analyze the algorithm for random graphs where $G \sim B(N,p)$. We will prove that if $p = \frac{c\log(N)}{N}$ and $c > 1$ then w.h.p the algorithm will terminate within $O(N\log(N))$ iterations. Furthermore, if $c > 2$ then the expected number of iterations would also be bounded by $O(N\log(N))$.

Definition 4. *$\tilde{B}(N,p)$ is a set of graphs where if $G \in \tilde{B}(N,p)$ then $G \sim B(N,p)$ and for any non-maximum matching M there exists an augmenting path for M of length at most $2L+1$ where $L = \frac{\tilde{c}\log(N)}{\log(Np)}$ and $\tilde{c} > 0$ is some constant.*

Observation 2. *Let $G \in \tilde{B}(N,p)$ and let and let T be the number of iterations until the algorithm terminates then $T \leq N(L+1)$ where $L = \frac{\tilde{c}\log(N)}{\log(Np)}$.*

Proof. $G \in \tilde{B}(N,p)$; hence, for any non maximum matching M there exists an augmenting path of length at most $2L+1$. let $L = \frac{\tilde{c}\log(N)}{\log(Np)}$, then by lemma 6 we get that $h_v(i) \leq L+1, \forall i, v \in V$. Using lemma 1 we get:

$$T = \sum_{v=1}^{N} h_v \leq N(L+1) = \frac{N\tilde{c}\log(N)}{\log(Np)} + N. \tag{10}$$

The following theorem was proven in [18]: ∎

Theorem 8. *Let $G \sim B(N,p)$ where $p \geq \frac{(1+\epsilon)\log(N)}{N}$ then for every $\epsilon, \gamma > 0$ there exists N_γ such that for every $N \geq N_\gamma$*

$$\Pr(G \in \tilde{B}(N,p)) \geq 1 - N^{-\gamma}. \tag{11}$$

The following theorem was proven in [10]:

Theorem 9. *Let $G = (U,V,E) \sim B(N,p)$ and $p = \frac{c\log(N)}{N}, c > 1$ then*

$$\lim_{N\to\infty} \Pr\,(G \textit{ contains a perfect matching}) = e^{-2N^{1-c}}. \tag{12}$$

We now prove the main theorem of the paper using the above known results:

Theorem 10. *Let $G \sim B(N,p)$ be a random bipartite graph with $N > N_\gamma$ vertices on each side and $p = \frac{c\log(N)}{N}, c > 2$. Then the expected number of iterations until the algorithm terminates is:*

$$E(T) \leq O\left(\frac{N\log(N)}{\log(\log(N))}\right) \tag{13}$$

Proof. Let

$$A = \left\{G \sim B\left(N, \frac{c\log(N)}{N}\right) : G \text{ does not contains a perfect matching}\right\} \tag{14}$$

and let

$$B = \left\{G \sim B(N, \frac{c\log(N)}{N}) \setminus \tilde{B}\left(N, \frac{c\log(N)}{N}\right)\right\}. \tag{15}$$

From Theorem 8 we know that for $\gamma > 1$ and for large enough N $\Pr(B) \leq N^{-\gamma}$. From theorem 9 we know that if $G \sim B\left(N, \frac{c\log(N)}{N}\right)$ and $c > 2$, then for large enough N $\Pr(A) \leq \frac{2}{N}$.

$$\breve{B}\left(N, \frac{c\log(N)}{N}\right) = \{G : G \text{ contains a perfect matching}\} \cap \tilde{B}\left(N, \frac{c\log(N)}{N}\right). \tag{16}$$

Using the union bound, for large enough N

$$\Pr\left(G \in \breve{B}\left(N, \frac{c\log(N)}{N}\right)\right) \geq 1 - \frac{3}{N}. \tag{17}$$

From lemma 7 and Lemma 2 we know that:

$$T \leq \begin{cases} N(N-1), & G \notin \breve{B}\left(N, \frac{c\log(N)}{N}\right) \\ \frac{N\tilde{c}\log(N)}{\log(\log(N))} + N, & G \in \breve{B}\left(N, \frac{c\log(N)}{N}\right) \end{cases}, \tag{18}$$

this implies that if $N \geq \tilde{N}$, $c > 2$ then

$$E(T) \leq \left(1 - \frac{3}{N}\right)\left(\frac{N\tilde{c}\log(N)}{\log(c + \log(N))} + N\right) + \frac{3}{N}(N^2 - N) = O\left(\frac{N\log(N)}{\log(\log(N))}\right). \quad (19)$$

∎

Note that we required $c > 2$ to make the second terms of the expectation at the order of less than $O(N\log(N))$. However, if we wish to get an upper bound on the worst case, then choosing $c > 1$ is sufficient.

5. EXPECTED TIME COMPLEXITY

In this section we use the results from the previous section to analyze the expected time complexity of the algorithm in sequential and parallel implementations. We first prove that the expected running time of the algorithm is $O\left(\frac{N\log^2(N)}{\log(Np)}\right)$. We point out that if the graph is dense it is advantageous to take a random sparse subgraph of the original graph and perform the algorithm on the sparse subgraph. Then, we introduce a parallel implementation of the algorithm for machines with $O(\log(N))$ processors and a shared memory. We prove that if the expected time complexity is $O(N\log(N))$.

5.1. Sequential Implementation

We now analyze the expected time complexity of the algorithm for random bipartite graphs from $B(N,p)$. From Theorem 10 we know that the expected number of iterations is bounded by $O\left(\frac{N\log(N)}{\log(Np)}\right)$. Hence we need to show that the expected number of operations per iteration is $O(\log(N))$. The next theorem shows that the algorithm can be implemented with $O\left(\frac{N\log^2(N)}{\log(Np)}\right)$ time complexity.

Theorem 11. *Let $G = (U,V,E) \sim B(N,p)$ be a random bipartite graph with $p \geq \frac{c\log(N)}{N}$, $c > 2$ then the algorithm finds a maximum matching on a sequential machine with one processor with the expected time bounded by $O\left(\frac{N\log^2(N)}{\log(Np)}\right)$*

Proof. In section 4 we proved that the expected number of iterations until convergence is bounded by

$$E(T) \leq O\left(\frac{N\log(N)}{\log(Np)}\right) \quad (20)$$

where T is the number of iterations until convergence. Define T_{inner} to be the number of operations needed in each iteration and T_{total} to be the total number of operations preformed by the algorithm.

All of the operations in each iterations are $O(1)$ except for the operation of finding a vertex with minimal value for the chosen vertex u which requires $O(|n_u^G|)$ operations where $|n_u^G|$ is the number of neighbours of vertex u. Since the number of neighbors of each vertex is an independent random variable and $E(|n_u^G|) = Np$ then

$$E(T_{total}) = E(T)\,E(T_{inner}) = O\left(\frac{pN^2\log(N)}{\log(Np)}\right) \quad (21)$$

If $E(|n_u^G|) = O(\log(N))$ then

$$E(T_{total}) = O\left(\frac{N\log^2(N)}{\log\log(N)}\right) \tag{22}$$

▪

When the graph is dense; i.e., $E(|n_u^G|) = O(N)$ the algorithm converges with a time complexity of $O\left(N^2\log(N)\right)$ which is not particularly good. To improve the expected running time performance, we can obtain a sparse random graph from the dense graph by randomly choosing d edges from the original dense graph where d is binomially distributed as

$$d \sim Bin\left(|E|, \frac{c\log(N)}{N}\right) \tag{23}$$

Let $\tilde{G}$ be the sparse graph obtained from G by randomly selecting d edges of G. If $\tilde{G}$ contains a perfect matching, a solution for the MCM problem for $\tilde{G}$ is also a solution for G. If $\tilde{G}$ doest not contain a perfect matching the algorithm is applied on G with $O\left(N^2\right)$ iterations. Hence, the expected time complexity even for dense graphs remains:

$$\begin{aligned} E(T_{total}) &\leq \left(1-\frac{2}{N}\right)O\left(\frac{N\log^2(N)}{\log\log(N)}\right) + \frac{2}{N}O\left(N^2\log(N)\right) \\ &= O\left(\frac{N\log^2(N)}{\log\log(N)}\right) \end{aligned} \tag{24}$$

5.2. Parallel Implementation

We now describe a parallel implementation of the algorithm on a machine with $O(\log(N))$ processors and a shared memory. We show that the expected time complexity of the algorithm is $O(N\log(N))$. As in the sequential case, the expected number of outer iterations is $T = O\left(\frac{N\log(N)}{\log(Np)}\right)$. This implies that for $p = \frac{c\log(N)}{N}$ $T = O\left(\frac{N\log(N)}{\log(\log(N))}\right)$. The main difference between the parallel implementation and the sequential implementation is that using $O(\log(N))$ processors allows us to keep sorted AVL trees for the neighbours of each $u \in U$. An AVL tree [1] is a search tree data structure that has the following properties: The search time as well as the insertion and deletion operations on an AVL tree with $O(M)$ nodes are $O(\log(M))$. On random graphs with $p = \frac{c\log(N)}{N}$ w.h.p for each vertex $u \in U$ and $v \in V$

$$\begin{aligned} |n_u^G| &= O(\log(N)), \quad \forall u \in U \\ |n_v^G| &= O(\log(N)), \quad \forall v \in V \end{aligned} \tag{25}$$

Hence, if $c > 2$ w.h.p on each iteration an expected number of $O(\log(N))$ AVL trees with $M = O(\log(N))$ nodes need to be changed. Since there are $O(\log(N))$ elements on each tree, the maintenance time of each tree is $O(\log(\log(N)))$ and if $O(\log(N))$ processors are used then each processor maintain $O(1)$ trees on each iteration. Also, the minimum value can be found on AVL trees with $O(\log(N))$ nodes with a time complexity of $O(\log(\log(N)))$. As a result, w.h.p each iteration runs on a machine with $O(\log(N))$ processors with a time complexity of $O(\log(\log(N)))$. and w.h.p the running time of the parallel algorithm is bounded by:

$$T_{total} = T \cdot T_{inner} \leq O\left(\frac{N\log(N)}{\log(\log(N))} \cdot \log(\log(N))\right) = O(N\log(N)) \tag{26}$$

TABLE 2. Parallel auction algorithm for maximum bipartite matching

Inputs: N the number of vertices on each side.
E a list of edges size $|E| \times 2$ where each line i represents an edge from $u_{E(i,1)}$ to $v_{E(i,2)}$
p original probability per edge
Q number of parallel processors

Initialization phase:
1. Make the graph sparse by sampling d random edges from E according to (23).
2. Initialize $h_v = 0, \forall v \in V$ and set $M = \emptyset$.
3. create a free vertex set $F = \{1...N\}$.
4. Create an empty list of neighbors $n^G_{v_j}$ for each vertex $v \in V$.
5. Create an empty avl tree $n^G_{u_j}$ for each vertex $v \in V$.
6. Forall $q = 1..Q$, i= 1 to $\frac{d-Q+q}{Q}$.
   6.1 $n^G_{v_{E(q\cdot i,2)}} = n^G_{v_{E(q\cdot i,2)}} \cup E(q \cdot i, 1)$.
   6.2 insert $E(q \cdot i, 2)$ into the $n^G_{u_{E(q\cdot i,1)}}$ tree.
7. return to 6.

Matching phase:
8. While $|M| < N$ and $\sum_{k=1}^{N} h_k < N(N-1)$ do
   8.1 Choose a free vertex $i \in F$
   8.2 $F = F \setminus u$
   8.3 $j = \arg\min_{v \in n^G_{u_i}} h_v$
   8.4 Forall processors $q = 1..Q$, $i = 1..\frac{|n^G_{v_j}|}{Q}$
      8.4.1 define $m = n^G_{v_j}$
      8.4.2 Delete j from $n^G_{u_m(q\cdot i)}$
      8.4.3 $h_j = h_j + 1$
      8.4.4 Insert j (with new value) into $n^G_{u_m(q\cdot i)}$
   8.5 return to 8.4.
   8.6 $u_{old} = \{u \in U : (u,j) \in M\}$
   8.7 $M = M \setminus (u_{old}, j)$
   8.8 $M = M \cup (u,j)$
9. return

The parallel algorithm for $O(\log(N))$ processors with a shared memory is given in Table 2.

6. CONCLUSION

In this paper we analyzed the expected time complexity of the auction algorithm for the matching problem on random bipartite graphs. We proved that the expected time complexity of the auction algorithm for bipartite matching is just as good as other augmenting path algorithms such as the HK algorithm. Furthermore, we showed that the algorithm can be implemented on parallel machines with $O(\log(N))$ processors and a shared memory with an expected time complexity of $O(N \log(N))$.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their detailed comments and suggestions for the manuscript. The authors believe that the comments have greatly improved the article.

REFERENCES

[1] M. AdelsonVelskii and E. M. Landis, An algorithm for the organization of information, Tech. report, DTIC Document, 1963.

[2] H. Bast, K. Mehlhorn, G. Schafer, and H. Tamaki, Matching algorithms are fast in sparse random graphs, Theory Comput Syst 39 (2006), 3–14.

[3] C. Berge, Two theorems in graph theory, Proc Nat Acad Sci USA 43 (1957), 842.

[4] D. Bertsekas, An auction algorithm for shortest paths, Massachusetts Institute of Technology, Laboratory for Information and Decision Systems, 1990.

[5] D. P. Bertsekas, A distributed algorithm for the assignment problem, Lab. for Information and Decision Systems Working Paper, MIT, 1979.

[6] D. P. Bertsekas and D. A. Castañon, A forward/reverse auction algorithm for asymmetric assignment problems, Comput Optim Appl 1 (1992), 277–297.

[7] P. Chebolu, A. Frieze, and P. Melsted, Finding a maximum matching in a sparse random graph in o (n) expected time, J ACM 57 (2010), 24.

[8] B. V. Cherkassky, A. V. Goldberg, P. Martin, J. Setubal, and J. Stolfi, Augment or push: a computational study of bipartite matching and unit-capacity flow algorithms, J Exp Algor 3 (1998), 8.

[9] E. A. Dinic, Algorithm for solution of a problem of maximum flow in networks with power estimation, Soviet Math. Dokl 11 (1970), 1277–1280.

[10] P. Erdős and A. Rényi, On the existence of a factor of degree one of a connected random graph, Acta Math Hungarica 17 (1966), 359–368.

[11] T. Feder and r. Motwani, Clique partitions, graph compression and speeding-up algorithms, Proceedings of the twenty-third annual ACM symposium on Theory of computing, ACM, 1991, pp. 123–133.

[12] A. Goel, M. Kapralov, and S. Khanna, Perfect matchings in o (n log n) time in regular bipartite graphs, Proceedings of the 42nd ACM symposium on Theory of computing, ACM, 2010, pp. 39–46.

[13] A. V. Goldberg and R. Kennedy, An efficient cost scaling algorithm for the assignment problem, Mathematical Programming 71 (1995), 153–177.

[14] A. V. Goldberg and R. E. Tarjan, A new approach to the maximum-flow problem, J ACM (1988), 921–940.

[15] J. E. Hopcroft and R. M. Karp, An n^5/2 algorithm for maximum matchings in bipartite graphs, SIAM J Comput 2 (1973), 225–231.

[16] K. Kaya, J. Langguth, F. Manne, and B. Uçar, Push-relabel based algorithms for the maximum transversal problem, Comput Oper Res (2012), 1266–1275.

[17] A. Madry, Navigating central path with electrical flows: from flows to matchings, and back, Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on, IEEE, 2013, pp. 253–262.

[18] R. Motwani, Average-case analysis of algorithms for matchings and related problems, J ACM 41 (1994), 1329–1356.

[19] J. Setubal, New experimental results for bipartite matching, Proc Netflow 93 (1993), 211–216.

[20] J. Setubal et al., Sequential and parallel experimental results with bipartite matching algorithms, University of Campinas, Tech. Rep. IC-96-09, 1996.