ELSEVIER

European Journal of Operational Research 104 (1998) 593–600

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Theory and Methodology

The β-assignment problems ☆

Gerard J. Chang [a,*], Pei-Hsin Ho [b]

[a] *Department of Applied Mathematics, National Chiao Tung University, Hsinchu 30050, Taiwan, ROC*

[b] *Intel Corporation, 2111 N.E. 25th Avenue, JFT-102, Hillsboro, OR 97124, USA*

Received 8 January 1993; accepted 8 October 1996

Abstract

Suppose $G = (S, T, E)$ is a bipartite graph, where (S, T) is a bipartition of the vertex set. A β-assignment is an edge set $X \subseteq E$ such that $\deg_X(i) = 1$ for all $i \in S$. The cardinality β-assignment problem is to find a β-assignment X which minimizes $\beta(X) = \max_{j \in T} \deg_X(j)$. Suppose we associate every edge with a weight which is a real number. The bottleneck β-assignment problem is to find a β-assignment X that minimizes $\beta(X)$ and maximizes the minimum edge weight on X. The weighted β-assignment problem is to find a β-assignment X that minimizes $\beta(X)$ and maximizes the total weights of edges in X. This paper presents $O(|S||E|)$-time algorithms for the cardinality and the bottleneck β-assignment problems and an $O(|S|^2|T| + |S||T|^2)$-time algorithm for the weighted β-assignment problem. © 1998 Elsevier Science B.V.

Keywords: Assignment; Bottleneck; Augmenting path; Label

1. Introduction

Chang and Lee [3] posed the following kind of assignment problem. Suppose there is a set S of n jobs and a set T of m workers. Information as to whether or not a worker is qualified for a job is known in advance. The problem is to assign jobs to workers such that the maximum number of jobs a worker has is minimized. To distinguish this problem from the traditional assignment problem [1,6–8], it is termed the *β-assignment problem*. This problem is formulated in terms of bipartite graphs as follows. Consider the bipartite graph $G = (S, T, E)$ in which (S, T) is a bipartition of the vertex set, and $(i, j) \in E$ if and only if worker j is qualified for job i. A *β-assignment* is an edge set $X \subseteq E$ such that

$$\deg_X(i) = 1 \quad \text{for all } i \in S,$$

where $\deg_X(i)$ is the degree of i in the subgraph of G induced by X. This implies that for any job i there exists exactly one worker j such that $(i, j) \in X$, and therefore job i is assigned to worker j. To apply β-assignments in scheduling problems, see [2].

Let $\beta(X)$ denote the maximum number of jobs a worker has in a β-assignment X, i.e.,

$$\beta(X) = \max_{j \in T} \deg_X(j).$$

The *cardinality β-assignment problem* is to find a β-assignment X which minimizes $\beta(X)$; this minimum value is denoted by $\beta(G)$. This study also takes account of the following variations of the cardinality β-assignment problem. In these variations, each edge (i, j) is associated with a weight w_{ij}, which can be interpreted as the profit accruing to a worker j by executing job i. The *bottleneck β-assignment problem*

☆ Supported in part by the National Science Council under grant NSC77-0208-M009-21.

* Corresponding author. E-mail: gjchang@math.nctu.edu.tw

PII S0377-2217(97)00008-8

is to find a β-assignment X with $\beta(X) = \beta(G)$ by which the minimum weight of an edge in X is maximized. The *weighted β-assignment problem* is to find a β-assignment X with $\beta(X) = \beta(G)$ by which the sum of the weights of all edges in X is maximized. Without loss of generality, it is assumed that G has a β-assignment, i.e., each vertex in S has a degree of at least one.

Chang and Lee [3] gave an $O(|S|^2|T|^2)$-time algorithm for the cardinality β-assignment problem. Chang [2] offered an $O(|S|^2|T|^2)$-time algorithm for the weighted β-assignment problem. This paper presents $O(|S||E|)$-time algorithms for the cardinality and the bottleneck β-assignment problems and an $O(|S|^2|T| + |S||T|^2)$-time algorithm for the weighted β-assignment problem. Strong duality theorems for these problems are incidentally verified.

2. The cardinality β-assignment problem

A *partial β-assignment* is an edge set $X \subseteq E$ such that $\deg_X(i) \leqslant 1$ for all $i \in S$. The proposed algorithm for the cardinality β-assignment problem starts with the empty partial β-assignment $X = \emptyset$ and adds one edge to X every iteration until an optimal β-assignment is found.

For a partial β-assignment X, a vertex i in S is *exposed* if $\deg_X(i) = 0$ and a vertex j in T is *safe* if $\deg_X(j) < \beta(X)$, otherwise it is *saturated.* If S' is the set of all non-exposed vertices in S, X also is termed a partial β-assignment of S'. An *X-alternating path* is a path whose edges are alternately in $E - X$ and X. An *X-augmenting path* is an X-alternating path whose origin is an exposed vertex in S and whose terminus a safe vertex in T.

The *symmetric difference* of two sets A and B is

$$A\Delta B = (A - B) \cup (B - A).$$

The following lemma is readily verified.

Lemma 2.1. *If X is a partial β-assignment of S' and P is an X-augmenting path starting at vertex $i \in S - S'$, then $X\Delta P$ is a partial β-assignment of $S' \cup \{i\}$ and $\beta(X\Delta P) = \beta(X)$.*

An *X-alternating tree* relative to a partial β-assignment X is a tree which is a subgraph of G and satisfies the following two conditions. First, the tree contains exactly one exposed vertex in S, which is called the *root* of the tree. Secondly, any path between the root and a vertex in the tree is an X-alternating path.

The proposed algorithm for the cardinality β-assignment problem begins with the empty partial β-assignment. Suppose the partial β-assignment X obtained so far is not a β-assignment. Then an exposed vertex s in S is located as the root of an X-alternating tree and vertices and edges are added to the tree by means of a labeling technique. Eventually, either a safe vertex in T is added to the X-alternating tree, or no further vertices or edges may be permitted. In the former case, an X-augmenting path is found and the partial β-assignment is augmented. In the latter case, all vertices in T of the X-alternating tree are saturated. Now, add an edge (s,t) to X; the value of $\beta(X)$ is increased by one. The tree-building procedure is repeated for $|S|$ iterations until an optimal β-assignment is obtained. More precisely, we obtain Algorithm Cardinality (see Fig. 1).

Algorithm Cardinality may be verified by employing the following dual problem of the cardinality β-assignment problem. For any $A \subseteq S$, $N_G(A)$ denotes the set of neighbors of A in graph G. In a β-assignment X, the vertices of A can be assigned only to vertices of $N_G(A)$, therefore

$$\beta(X) \geqslant \lceil |A|/|N_G(A)| \rceil$$

by the pigeonhole principle. Consequently, the following min-max duality inequality obtains.

Lemma 2.2.

$$\min_{X:\beta-\text{assignment}} \beta(X) \geqslant \max_{A\subseteq S} \lceil |A|/|N_G(A)| \rceil.$$

Theorem 2.3. *Algorithm* Cardinality *works.*

Proof. Since X is updated only in Steps (a) and (b), it continues to serve as a partial β-assignment by Lemma 2.1 and the definition. After $|S|$ iterations, the partial β-assignment becomes a β-assignment. Let X^* be the final β-assignment and k^* the final k obtained from the algorithm. Suppose L is the X-alternating tree rooted at s that forces the value of k to increase from $k^* - 1$ to k^* in Step (b), where X is a partial β-assignment of some A that does not contain s. $N_G(L \cap S) = L \cap T$ by the labeling method in case 1.

Algorithm Cardinality
Input: A bipartite graph $G = (S, T, E)$ with bipartition (S, T).
Output: An optimal cardinality β-assignment X with $\beta(X) = \beta(G)$.

```
X ← ∅;
k ← 0; {* where k = β(X) at any time *}
for each s ∈ S do
      set all vertices 'unscanned';
      erase labels of all vertices;
      label s by '∅';
(*)   if there is an unscanned and labeled vertex i
      then {scan i in the following three cases;
            case 1. i ∈ S {* the tree grows from vertices of S to T*}:
                  label each unlabeled j ∈ T adjacent to i by 'i';
                  goto (*);
            case 2. i ∈ T and is saturated {* the tree grows from T to S*}:
                  identify the k edges (i, j_1), (i, j_2), ..., (i, j_k) of X;
                  label each j_p by 'i' for 1 ≤ p ≤ k;
                  goto (*);
(a)         case 3. i ∈ T and is safe {* an X-augmenting path found *}:
                  backtrack from i to s by labels to get an X-augmenting path P;
                  X ← XΔP;}
(b)   else {* all vertices in T of the X-alternating tree are saturated *}
            {choose an edge e = (s, t);
            X ← X + e;
            k ← k + 1;}
      endif;
endfor;
output (X, k);
end Cardinality
```

Fig. 1. Algorithm Cardinality.

By Step (b), all the vertices of T in the X-alternating tree L are saturated, i.e., $\deg_X(j) = k^* - 1$ for all $j \in N_G(L \cap T)$. Also, $N_{G[X]}(L \cap T) = (L \cap S) - \{s\}$. Let $A^* = L \cap S$. Then, $|A^*| = |L \cap S| = (k^* - 1)|L \cap T| + 1 = (k^* - 1)|N_G(A^*)| + 1$, and therefore $\beta(X^*) = k^* = \lceil |A^*|/|N_G(A^*)| \rceil$. This, together with Lemma 2.2, gives

$$\begin{aligned}\beta(X^*) &\geqslant \min_{X:\beta-\text{assignment}} \beta(X)\\ &\geqslant \max_{A\subseteq S} \lceil |A|/|N_G(A)| \rceil\\ &\geqslant \lceil |A^*|/|N_G(A^*)| \rceil = \beta(X^*).\end{aligned}$$

Hence, all inequalities are in fact equalities. This verifies that X^* is an optimal β-assignment and the algorithm is therefore valid. □

Corollary 2.4.

$$\min_{X:\beta-\text{assignment}} \beta(X) = \max_{A\subseteq S} \lceil |A|/|N_G(A)| \rceil.$$

Corollary 2.4. is an equivalent statement of Edmonds and Fulkerson's theorem [4]. Note that the complexity of each iteration in the algorithm is $O(|E|)$, since constructing of an alternating tree utilizes at most $|E|$ edges and augmenting the assignment requires $O(|S|)$ time. Hence, the time complexity of the algorithm is $O(|S||E|)$.

3. The bottleneck β-assignment problem

Recall that the bottleneck β-assignment problem is to find a β-assignment X with $\beta(X) = \beta(G)$ that maximizes $\min\{w_{ij} : (i, j) \in X\}$. The algo-

```
Algorithm Bottleneck
Input: A bipartite graph G = (S,T,E) and a weight w_ij for each edge (i,j) ∈ E.
Output: An optimal bottleneck β-assignment X.
call Cardinality (G); {* to get k* = β(G) *}
X ← ∅;
W ← ∞; {* where W is the threshold *}
for each s ∈ S do
        set all vertices 'unscanned';
        erase labels of all vertices;
        labels s by '∅';
        π_j ← -∞ for all j ∈ T;
(*)    if each unscanned and labeled vertex j is in T and π_j < W
        then W ← max{π_j : π_j < W}; {* reduce the threshold *}
        select an unscanned and labeled vertex i in S or in T with π_i ≥ W;
        scan i for three cases;
                case 1. i ∈ S:
                        for each j ∈ T with (i,j) ∈ E - X, π_j < w_ij and π_j < W,
                                label j by 'i' and π_j ← w_ij;
                        goto (*);
                case 2. i ∈ T and is saturated, i.e., deg_X(i) = k*:
                        identify the k* edges (i,j_1),(i,j_2),...,(i,j_k*) of X;
                        label each j_p by 'i' for 1 ≤ p ≤ k*;
                        goto (*);
                case 3. i ∈ T and is safe, i.e., deg_X(i) < k*:
                        backtrack from i to s by labels to get an X-augmenting path P;
                        X ← XΔP;}
        endcase;
endfor;
output (X,W);
end Bottleneck
```

Fig. 2. Algorithm Bottleneck.

rithm introduced here starts with the empty partial β-assignment and a suitable large *threshold* W. Suppose that a partial β-assignment X of some $S' \subseteq S$ has been obtained at the general step. One tries to find an X-augmenting path in the subgraph containing all arcs (i,j) for which $w_{ij} \geqslant W$. To do this efficiently, a number π_j is associated with each vertex $j \in T$ such that

$$\pi_j = \max\left\{w_{ij} : (i,j) \in E \text{ and } i \text{ is in the } X\text{-alternating tree}\right\}.$$

While growing the X-alternating tree, vertices are labeled but no labeled vertex j in T is scanned unless $\pi_j \geqslant W$. If augmentation is possible, a partial β-assignment of $S' \cup \{s\}$ results, where whether a vertex of T is safe or not is determined by $\beta(G) = k^*$, which is obtained from algorithm Cardinality. If augmentation is not possible, the threshold W is reduced to the maximum value of π_j strictly less than W. This permits adding at least one vertex to the tree. Eventually, augmentation must occur, or otherwise by an argument similar to Theorem 2.3, $\beta(G) > k^*$, which is a contradiction.

The precise algorithm, Algorithm Bottleneck, is given in Fig.2.

For the reasons adduced in the cardinality case, the time complexity of Algorithm Bottleneck is also $O(|S||E|)$. The algorithm is verified again by a primal–dual approach. Let H denote a subgraph obtained from G by deleting p vertices of S and q vertices of T such that

$$p + \beta(G)q = |S| - 1.$$

Suppose X is a β-assignment with $\beta(X) = \beta(G)$. X has at most $\beta(G)q$ edges incident to the q deleted vertices of T and p edges incident to the p deleted vertices of S. Thus, H contains at least one of the $|S|$ edges of X. Therefore, the following lemma obtains.

Lemma 3.1.

$$\max_{X:\beta(X)=\beta(G)} \min_{(i,j)\in X} w_{ij} \leqslant \min_{H} \max_{(i,j)\in H} w_{ij}.$$

Theorem 3.2. *Algorithm* Bottleneck *works.*

Proof. Let X^* be the final β-assignment obtained by algorithm Bottleneck. Suppose the augmentation from a partial β-assignment X' of A to a partial β-assignment X of $A \cup \{s\}$ is the first time an edge $e_0 = (i_0, j_0)$ with the minimum weight in X^* is included by the assignment. Let L be the set of labeled vertices of G while the X'-alternating tree cannot extend further and that causes the reduction of the threshold W to $w(e_0)$.

Let $T_1 = \{j \in L \cap T : \deg_{G[L\cap V(G[X'])]}(j) \geqslant \beta(G)\}$ and H be the subgraph of G obtained by deleting the vertices of $(S - L) \cup T_1$. Since $|T_1| = (|L \cap S| - |\{s\}|)/\beta(G)$, then $|S - L| + \beta(G)|T_1| = |S - L| + |L \cap S| - 1 = |S| - 1$. It follows from Lemma 3.1 that $\min\{w_{ij} : (i,j) \in X^*\} \leqslant \max\{w_{ij} : (i,j) \in H\}$. Since $e_0 \in X$ and $w(e_0) = \min\{w_{ij} : (i,j) \in X^*\}$, it suffices to declare that $e_0 \in H$ and $w(e_0) = \max\{w_{ij} : (i,j) \in H\}$.

Note that $V(H) = (L \cap S) \cup T_2$, where $T_2 = T - T_1$. Furthermore, $i_0 \in L \cap S$ and $j_0 \in T_2$, so $e_0 \in H$. Since e_0 is the first bottleneck included by X^*, threshold W must be greater than $w(e_0)$ before $e_0 \in X^*$. By the choice of e_0, $\pi_{j_0} = \max\{\pi_j : \pi_j < W$ and j is an unscanned but labeled vertex of $T\}$ and $w(e_0) = \max\{w_{ij} : (i, j_0) \in E$ and i is a labeled vertex in $S\} = \pi_{j_0}$. Because T_2 is the set of the unscanned labeled vertices of T and $L \cap S$ is the set of the labeled vertices of S, it follows that $w(e_0) = \max\{w_{ij} : (i,j) \in H\}$. □

Corollary 3.3.

$$\max_{X:\beta(X)=\beta(G)} \min_{(i,j)\in X} w_{ij} = \min_{H} \max_{(i,j)\in H} w_{ij}.$$

4. The weighted β-assignment problem

The weighted β-assignment problem is to find a β-assignment X with $\beta(X) = \beta(G)$ which maximizes the total weights of the edges in X. Suppose $k^* = \beta(G)$ is obtained by the cardinality β-assignment algorithm. The proposed procedure for the weighted β-assignment problem is a primal-dual method. The integer linear programming formulation of the weighted β-assignment problem is:

$$\text{Maximize} \quad \sum_{(i,j)\in E} w_{ij}x_{ij}$$

$$\text{subject to} \quad \sum_{j\in T} x_{ij} = 1 \quad \text{for all } i \in S, \tag{1}$$

$$\sum_{i\in S} x_{ij} \leqslant k^* \quad \text{for all } j \in T, \tag{2}$$

$$x_{ij} \geqslant 0 \quad \text{for all } (i,j) \in E, \tag{3}$$

$$x_{ij} \text{ integer} \quad \text{for all } (i,j) \in E. \tag{4}$$

Note that condition (4) can be replaced by 'x_{ij} is binary for all $(i,j) \in E$'. A feasible solution $(x_{ij} : (i,j) \in E)$ is equivalent to a β-assignment $X = \{(i,j) \in E : x_{ij} = 1\}$. The *dual* of its *linear programming relaxation* (i.e. (4) is dispensed with) is:

$$\text{Minimize} \quad \sum_{i\in S} u_i + k^* \sum_{j\in T} v_j$$

$$\text{subject to} \quad v_j \geqslant 0 \quad \text{for all } j \in T, \tag{5}$$

$$u_i + v_j \geqslant w_{ij} \quad \text{for all } (i,j) \in E. \tag{6}$$

The *orthogonality conditions* are

$$\left(\sum_{i\in S} x_{ij} - k^*\right) v_j = 0 \quad \text{for all } j \in T, \tag{7}$$

$$x_{ij}(u_i + v_j - w_{ij}) = 0 \quad \text{for all } (i,j) \in E. \tag{8}$$

By linear programming theory, solutions of the primal and the dual problems are optimal if and only if they satisfy conditions (1)–(8). The weighted β-assignment problem algorithm offers initial solutions that satisfy all conditions except (1). The number of vertices $i \in S$ such that condition (1) fails decreases by one for each iteration of the algorithm (see Algorithm Weight, Fig. 3).

The procedure begins with the empty partial β-assignment $X = \emptyset$ and the dual solution $u_i =$

```
Algorithm Weight
Input: A bipartite graph G = (S,T,E) and a weight w_ij for each edge (i,j) ∈ E.
Output: An optimal weighted β-assignment X.
call Cardinality (G); {* to get k* = β(G) *}
X ← ∅;
u_i ← max_{j∈T} w_ij for all i ∈ S;
v_j ← 0 for all j ∈ T;
for each s ∈ S do
      erase labels of all vertices;
      label s by '∅';
      π_j ← ∞ for all j ∈ T;
(*)   if there is an unscanned but labeled vertex i ∈ S or i ∈ T with π_i = 0
      then {scan i in the following three cases;
            case 1. i ∈ S:
                  for each j ∈ T with (i,j) ∈ E and u_i + v_j − w_ij < π_j,
                        label j by 'i' and π_j ← u_i + v_j − w_ij;
                  goto (*);
            case 2. i ∈ T and is saturated, i.e., deg_X(i) = k*:
                  identify the k* edges (i,j_1), (i,j_2), ..., (i,j_k*) of X;
                  label each j_p by 'i' for 1 ≤ p ≤ k*;
                  goto (*);
            case 3. i ∈ T and is safe, i.e., deg_X(i) < k*:
                  backtrack from i to s by labels to get an X-augmenting path P;
                  X ← XΔP;}
      else {δ ← min{π_j : π_j > 0 and j ∈ T};
            u_i ← u_i − δ for all labeled i ∈ S;
            v_j ← v_j + δ for all j ∈ T with π_j = 0;
            π_j ← π_j − δ for all j ∈ T with π_j > 0;
            goto (*);}
      endif; endfor;
output (X, Σ_{ij∈X} w_ij);
end Weight
```

Fig. 3. Algorithm Weight.

$\max_{j\in T} w_{ij}$ for all $i \in S$ and $v_j = 0$ for all $j \in T$. These initial primal and dual solutions clearly satisfy conditions (2)–(8). At the general step of the procedure, conditions (2)–(8) hold, but for some $i \in S$, condition (1) does not. Then, by a labeling method, an augmenting path is sought within the subgraph containing only edges (i,j) for which $u_i + v_j = w_{ij}$, so as to ensure continuing to satisfy condition (8). If such a path P is found, then X is updated by $X\Delta P$. The new partial β-assignment continues to meet conditions (2)–(8) and the number of vertices $i \in S$ such that condition (1) fails decreases by one. If augmentation is not possible, then all the edges (i,j) available for continual addition to the X-alternating tree are such that $u_i + v_j > w_{ij}$. Such edges are incident to a vertex of S in the X-alternating tree and a vertex of T that is not so. Then, a change of certain 'suitable' $\delta > 0$ is made in the dual variables by subtracting δ from u_i for each tree vertex $i \in S$ and adding δ to v_j to each tree vertex $j \in T$. Such a change in the dual variables affects the net value of $u_i + v_j$ only for edges that have one end in the tree and the other end not so. The authors contend that after such a change, the new dual variables continue to satisfy conditions (2)–(8). Note that only conditions (5)–(8) require checking. Condition (5) remains true since the new value of

Table 1

$\lvert S\rvert$	$\lvert T\rvert$	ρ	ρ'	$\beta(G)$	CPU (sec) for $\beta(G)$	$b(G)$	CPU (sec) for $b(G)$	$w(G)$	CPU (sec) for $w(G)$
150	30	0.010	0.035	10	0.033332	10	0.033332	8031	0.016666
150	30	0.320	0.322	5	0.033332	47	0.033332	11304	0.049998
300	30	0.010	0.034	14	0.049998	10	0.116662	16562	0.066664
300	30	0.320	0.320	10	0.099996	37	0.166660	24336	0.099996
300	60	0.010	0.019	10	0.066664	10	0.133328	17584	0.066664
300	60	0.320	0.324	5	0.116662	75	0.183326	25227	0.116662
450	30	0.010	0.035	25	0.133328	10	0.233324	25106	0.133328
450	30	0.320	0.328	15	0.216658	35	0.349986	38085	0.183326
450	60	0.010	0.020	13	0.133328	10	0.266656	25894	0.149994
450	60	0.320	0.319	8	0.233324	71	0.349986	39482	0.216658
450	90	0.010	0.014	8	0.149994	10	0.283322	26059	0.149994
450	90	0.320	0.319	5	0.249990	75	0.399984	37548	0.299988
600	30	0.010	0.034	27	0.216658	10	0.433316	33257	0.233324
600	30	0.320	0.321	20	0.383318	34	0.616642	50030	0.299988
600	60	0.010	0.019	15	0.233324	10	0.433316	33416	0.266656
600	60	0.320	0.318	10	0.449982	68	0.583310	52553	0.366652
600	90	0.010	0.014	12	0.233324	10	0.466648	34607	0.266656
600	90	0.320	0.321	7	0.416650	77	0.599976	52777	0.433316
600	120	0.010	0.012	9	0.266656	10	0.483314	35208	0.299988
600	120	0.320	0.320	5	0.433316	80	0.649974	52377	0.499980
750	30	0.010	0.035	43	0.316654	10	0.666640	18694	0.349986
750	30	0.320	0.319	25	0.599976	32	0.933296	31936	0.416650
750	60	0.010	0.019	22	0.349986	10	0.683306	17426	0.366652
750	60	0.320	0.320	13	0.599976	69	0.833300	33310	0.483314
750	90	0.010	0.015	12	0.349986	10	0.733304	26447	0.416650
750	90	0.320	0.321	9	0.649974	77	0.883298	44392	0.533312
750	120	0.010	0.013	11	0.366652	10	0.783302	32159	0.466648
750	120	0.320	0.321	7	0.783302	82	0.949962	51087	0.616642
750	150	0.010	0.012	7	0.399984	10	0.783302	34543	0.499980
750	150	0.320	0.320	5	0.733304	89	1.066624	50773	0.733304

each v_j is greater than or equal to its old value. The only case for decreasing u_i+v_j is when i is a tree vertex but j not. In that event u_i+v_j is decreased by δ. Since originally $u_i+v_j > w_{ij}$, selecting a sufficiently small δ can make (6) true. The only opportunity for increasing v_j from zero to δ occurs when j is a tree vertex. But each tree vertex $j \in T$ has the property that $\sum_{i\in S} x_{ij} = k^*$, and condition (7) still holds. The only case of $u_i + v_j - w_{ij}$ changing from zero to δ is when j is a tree vertex but i is not. By cases 2 and 3 of the algorithm, $(i,j) \notin X$ or $x_{ij} = 0$, so condition (8) still holds. Therefore, after the change, the dual variables continue to satisfy conditions (2)-(8).

As is the case for the threshold algorithm for the bottleneck optimal assignment problem, a number π_j is associated with each vertex j in T. This number indicates the value of δ so that j may be added to the tree. The labeling procedure progressively decreases π_j until $\pi_j = \min\{u_i+v_j-w_{ij} : (i,j) \in E \text{ and } i \in S$ is in the alternating tree$\}$. Note that a vertex $j \in T$ may receive a label although $\pi_j > 0$ but its label is scanned only if $\pi_j = 0$. Since we let $\delta = \min\{\pi_j : \pi_j > 0$ and $j \in T\}$ in the algorithm, at least one new edge can be added to the tree provided that G has a β-assignment. Thus, the X-alternating tree continues to grow.

After $\lvert S\rvert$ iterations, the resulting β-assignment satisfies conditions (1)-(8) and therefore is optimal. In each sub-iteration of an iteration, the algorithm either scans a vertex or modifies the dual variables. Note that no vertex is scanned more than once in the same iteration; and after modifying dual variables, a labeled vertex always remains to be scanned. Therefore, there are at most $|T|$ dual variable modifications in an iteration. Since each modification costs $O(|S| + |T|)$ operations, each iteration requires $O(|S||T| + |T|^2)$ operations for the dual variable modifications. Because constructing the X-alternating tree employs at most $|E| \leqslant |S||T|$ edges, the time complexity of this algorithm is $O(|S|^2|T| + |S||T|^2)$.

If either a max-flow-like or a shortest-path-like procedure is utilized to determine maximum weighted augmentation at each iteration, the time complexity of the algorithm is $O(|S|^3|T|)$.

5. Numerical results

The three algorithms of this paper were coded in a C program and run on a SUN SPARC 10. Bipartite graphs of various size were generated with two kinds of edge densities. Table 1 illustrates a typical output of the C program.

The first (second) column is the size of S (T). The third column is the probability ρ for the existence of an edge ij. A random number generator determines whether or not ij is an edge. To ascertain that $\beta(G)$ exists, when a vertex i has degree zero, an edge ij is randomly added to the graph, thus rendering the real edge density $\rho' = |E|/|S||T|$, as is depicted in the fourth column, larger than ρ for some cases. Column 5 indicates the value $\beta(G)$ obtained from algorithm Cardinality and column 6 the running time. Column 7 is the maximum value $b(G)$ of the minimum weight of an edge in a β-assignment X with $\beta(X) = \beta(G)$ and column 8 the running time. Column 9 is the maximum value $w(G)$ of the sum of the weights of all edges in a β-assignment X with $\beta(X) = \beta(G)$ and column 10 the running time.

Acknowledgements

The authors thank the referees for many constructive suggestions for the revision of the paper.

References

[1] M.L. Balinski, R.E. Gomory, A primal method for the assignment and transportation problems, Management Science 10 (1964) 578–593.

[2] R.S. Chang, A weighted job assignment problem and #P-complete result, in: Proceedings of the 3rd International Conference for Young Computer Scientists, Beijing, China, July, 1993, pp. 15–17.

[3] R.S. Chang, R.C.T. Lee, On a scheduling problem where a job can be executed only by a limited number of processors, Computers & Operations Research 15 (1988) 471–478.

[4] J. Edmonds, D.R. Fulkerson, Minimum partition of a matroid into independent subsets, Journal of Research of the National Bureau of Standards 69B (1965) 67–72.

[5] J. Edmonds, D.R. Fulkerson, Bottleneck extrema, Journal of Combinatorial Theory 8 (1970) 299–306.

[6] O. Gross, The bottleneck assignment problem: an algorithm, in: P. Wolfe (Ed.), Proceedings, Rand Symposium on Mathematical Programming, Rand Publication R-351, 1960, pp. 87–88.

[7] H.W. Kuhn, The Hungarian method for the assignment problems, Naval Research Logistics Quarterly 2 (1955) 83–97.

[8] H.W. Kuhn, Variants of the Hungarian methods for assignment problems, Naval Research Logistics Quarterly 3 (1956) 235–258.

[9] E.L. Lawler, Combinatorial Optimization: Networks and Matroids, Holt, Rinehart & Winston, New York, 1976.

[10] C.H. Papadimitriou, K. Steiglitz, Combinational Optimization: Algorithms and Complexity, Prentice-Hall, Englewood Cliffs, NJ, 1982.