Available online at www.sciencedirect.com
ScienceDirect
Discrete Optimization 5 (2008) 605–614
DISCRETE OPTIMIZATION
www.elsevier.com/locate/disopt

Worst case examples of an exterior point algorithm for the assignment problem

Charalampos Papamanthou[a], Konstantinos Paparrizos[b], Nikolaos Samaras[b,*], Konstantinos Stergiou[c]

[a] *Department of Computer Science, Brown University, Providence RI, USA*
[b] *Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece*
[c] *Department of Information and Communication Systems Engineering, University of the Aegean, Samos, Greece*

Received 9 December 2005; received in revised form 12 October 2007; accepted 19 December 2007
Available online 20 February 2008

Abstract

An efficient exterior point simplex type algorithm for the assignment problem has been developed by Paparrizos [K. Paparrizos, An infeasible (exterior point) simplex algorithm for assignment problems, Math. Program. 51 (1991) 45–54]. This algorithm belongs to the category of forest algorithms and solves an $n \times n$ assignment problem in at most $\frac{n(n-1)}{2}$ iterations and in at most $O(n^3)$ time. In this paper worst case examples are presented. Specifically, a systematic procedure to construct worst case assignment problems is presented for the exterior point algorithm. The algorithm applied to these examples executes exactly $\frac{n(n-1)}{2}$ iterations. This result verifies that the bound $O(n^3)$ is the best possible for the above-mentioned algorithm.

MSC: 90C08; 90C27; 90C35

Keywords: Assignment problem; Exterior point algorithm; Worst case examples

1. Introduction

The Assignment Problem (AP) is one of the well-studied problems in the field of Computer Science and Operations Research. A lot of algorithms have been developed for its solution [1]. The worst case complexity of the best algorithms for the AP is $O(n^3)$, where n is the size of the problem. The main algorithm categories for the AP are the primal-dual, the simplex type, the cost operator, the recursive, the forest and the interior point algorithms. Surveys on methods which solve the AP have been presented by Akgül [2], Derigs [3] and Martello and Toth [4]. Using the Hirsch-conjecture for dual transportation polyhedra, Balinski [5] developed a signature method for the dual AP [6,7]. An extension of Balinski's method to other transportation problems has been presented by Kleinschmidt et al. [8]. An efficient category of simplex type algorithms for the AP [9–11] has been developed. This category is called Exterior

* Corresponding address: 156 Egnatia Str., 54006 Thessaloniki, Greece. Tel.: +30 2310 891866; fax: +30 2310 891879.
E-mail addresses: cpap@cs.brown.edu (C. Papamanthou), paparriz@uom.gr (K. Paparrizos), samaras@uom.gr (N. Samaras), konsterg@aegean.gr (K. Stergiou).

1572-5286/$ - see front matter
doi:10.1016/j.disopt.2007.12.003

Point Simplex Algorithms (EPSAs). These simplex type algorithms are modifications of the classical network simplex algorithm. Roughly speaking they are initialized with a feasible tree or forest. During the computations the feasibility is destroyed and is restored again at optimality. According to the nature of the initial basic solution, simplex type algorithms can be further divided into two classes; primal and dual simplex type algorithms.

Numerical experiments are an indispensable part of research in Discrete Optimization. One of the difficulties in designing and executing experiments is finding appropriate test problems which reveal extremes of behavior. Why is this category of test problems important? There are two reasons which explain the necessity of these problems; (i) to verify the correctness of a complexity analysis and to check if the analysis completely explains the practical behavior and (ii) to gain insight into accuracy, reliability and computational effort.

The aim of this paper is to mathematically define test problems, independently of size, which can force the EPSA [11] to perform maximum iterations. In [11] it is proved that the algorithm is bound to perform at most $\frac{n(n-1)}{2}$ iterations when solving an AP of size n. The overall complexity of this algorithm is $O(n^3)$. This algorithm belongs to the special class of forest algorithms. EPSA is initialized with a basic dual feasible tree and constructs a spanning forest which is dual feasible. The input data of an AP is an $n \times n$ square matrix c. Therefore we have to define a square matrix c as a function of n. Little literature exists on similar efforts. Balinski [5] has proved that an exterior point algorithm with a different starting solution, when applied to the transportation problem of size $m \times n$, will always perform the maximum iterations if $c_{ij} = (m - i) \times (j - 1)$, $i = 1, \ldots, m$, $j = 1, \ldots, n$. Our examples are modifications of Balinski's examples. Computational results confirm that the test problems generated using our method are hard not only for the Dual Forest Exterior Point Algorithm of [11] but also for all the exterior point simplex algorithms.

We begin in Section 2 with the mathematical formulation of the assignment problem and a short description of the algorithm described in [11]. In Section 3 some basic methodology and notations are presented. In Section 4, we prove the correctness of the proposed generation scheme. Computational results are presented in Section 5. Finally, Section 6, contains our conclusions.

2. Basic definitions

2.1. Mathematical formulation

The mathematical formulation of the AP with a square $n \times n$ cost matrix c is the following:

$$\text{(P)} \ \min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{1}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} x_{ij} = 1, \ j = 1, \ldots, n \tag{2}$$

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1, \ldots, n \tag{3}$$

$$0 \leq x_{ij}, 1 \leq i, j \leq n. \tag{4}$$

Problem (P) can be solved as a linear program. All x_{ij}'s of the optimal solution are integral, that is 0-1.The associated dual problem (D) is

$$\text{(D)} \ \max \left(\sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j \right)$$

$$\text{s.t.} \quad u_i + v_j \leq c_{ij}$$

$$1 \leq i, j \leq n.$$

The reduced costs s_{ij} can now easily be computed, using the following relation

$$s_{ij} = c_{ij} - u_i - v_j. \tag{5}$$

Given a pair of solutions x and (u, v) for the problems (P) and (D) respectively, the optimality conditions (or complementary slackness) are

$$x_{ij}s_{ij} = 0, \quad 1 \leq i, j \leq n.$$

2.2. The Dual Forest Algorithm

It is well known that an AP can be represented by a bipartite graph of the form $G(S, D, E) = G(N, E)$, which consists of two discrete sets of nodes S, D ($N = S \cup D$) such as $|S| = |D| = n$. Here, E is the set of arcs directed from nodes of S to nodes of D. Nodes $i \in S$ are called supply nodes, whereas nodes $j \in D$ are called demand nodes. In our pictures we draw supply nodes as circles and demand nodes as squares. The main idea of the dual forest algorithm [11] is as follows. At each iteration a solution T is computed. Structure T is depicted as a forest of the bipartite graph. Using the current solution T, the set of nodes are partitioned into two subsets F^S and $F^D = T \sim F^S$. The algorithm stops when F^S becomes void. The algorithm is of simplex type, and hence an entering arc (g, h) and a leaving arc (k, l) are chosen at each iteration. First, the entering arc (g, h) is chosen by the relation

$$s_{gh} = \delta = \min\{s_{ij} : i \in F^S \wedge j \in F^D\}. \tag{6}$$

Then the leaving arc (k, l) is chosen according to the degree d of node h. If h is not an isolated node (i.e., $d(h) > 0$), then (k, l) is the sole downward arc of the tree (i, h). Else, if h is an isolated node, then the isolated nodes of the tree are decremented [11] (it is like removing an artificial arc between the isolated node and an artificial node 0).

The algorithm is initialized with a solution (forest) computed as follows. Let T denote this forest. Forest T consists of n subtrees T_j. Let $u_i(T)$, $v_j(T)$, $i, j = 1, 2, \ldots, n$ denote the dual variables that correspond to the supply and the demand nodes of the forest respectively. Initially we set $v_j(T) = 0$, $\forall j \in D$. In succession, we compute the dual variables of the supply nodes $u_i(T)$ by setting

$$v_j(T) = \min\{c_{ij} : j = 1, \ldots, n\}, i \in S \tag{7}$$

Let t be the column node such that $u_i(T) = c_{it}$. Then, the arc (i, t) is a basic arc of the forest T. For each arc (i, t) we set $x_{it} = 1$. The reduced costs s_{ij} can now easily be computed, using relation (5). Note that $s_{ij}(T) \geq 0$, which means that forest T is dual feasible. The initial set computed using the relation $F^S = \{T_j : d(j) \geq 1\}$. Accordingly, $F^D = \{T_j : d(j) < 1\}$. A general initialization forest for the dual forest algorithm is depicted in Fig. 1. One special data structure that the algorithm uses is the tree T^*. This tree contains the nodes that are "cut off" the current tree when the leaving arc is discarded. Only the reduced costs of arcs with one of its nodes belonging to the tree T^* and the other to the subtree $T \sim T^*$ are updated. Algorithm 1 is the pseudocode of the described algorithm.

Algorithm 1 DUAL_FOREST(C)

```
1: Compute F^S, F^D, the reduced costs s_ij and the initial solution T;
2: while F^S ≠ Ø do
3:    choose entering arc (g, h) by s_gh = δ = min{s_ij : i ∈ F^S ∧ j ∈ F^D};
4:    choose leaving arc (k, l) according to the degree of node h [11];
5:    T' = T ∪ (g, h) ∼ (k, l);
6:    if h ∈ T* then
7:       q = −δ;
8:    else
9:       q = δ;
10:   end if
11:   for each row node i ∈ T* set s_i.(T') = s_i.(T) − qe^T;
12:   for each column node j ∈ T* set s_.j(T') = s_.j(T) + qe;
13:   update F^S, F^D;
14:   T = T';
15: end while
```

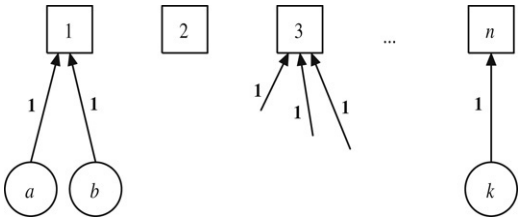

Fig. 1. A general initialization forest for the dual forest algorithm. $T_1, T_3, T_n \in F^S$ and $T_2 \in F^D$.

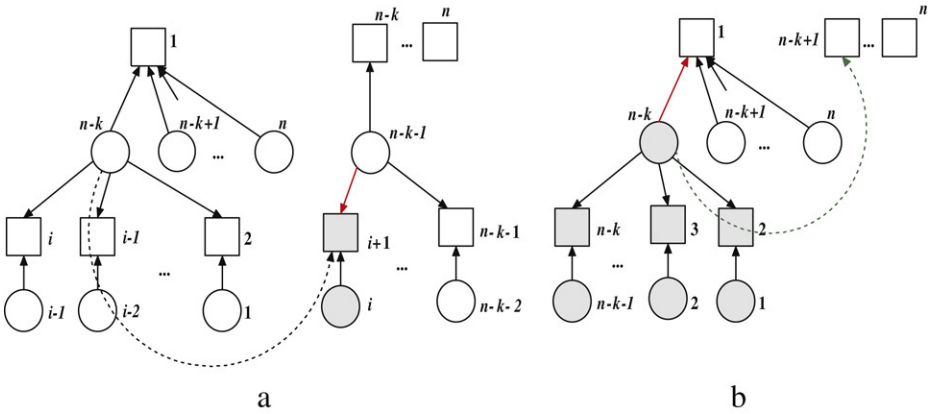

Fig. 2. (a) Stage k, Stage Iteration i, $i < n - k$, (b) Stage k, Stage Iteration i, $i = n - k$.

3. Methodology

As it is already known [11], the described algorithm can be divided into stages. The algorithm, before reaching the optimal solution, will pass through a number of stages equal to the number of isolated nodes of the initial solution (the stages are numbered from $n - 1$ to 0). The maximum number of stages is $n - 1$, where n is the size of the AP. Hence the algorithm will pass through all $n - 1$ stages if and only if the initial solution (i.e., the initial forest) will contain exactly $n - 1$ isolated nodes. This means that a basic precondition to achieve the maximum number of iterations is the existence of n arcs (i, w), $i = 1, \ldots, n$, where w is an arbitrarily chosen column node. In our case $w = 1$. This can be achieved when

$$c_{i1} = \min\{c_{ij} : j = 1, \ldots, n\}, \quad i = 1, \ldots, n.$$

Initially, the algorithm is at stage $n - 1$. In [11], it is proved that when the algorithm is at stage k, it executes at most $n - k$ iterations. Thus we need to provide our algorithm with data, so that this bound can be achieved for every stage. The passing from one stage to the other takes place when $d(h) = 0$ (i.e., h is an isolated node), where (g, h) is the entering arc. So we have to ensure that at the first $n - k - 1$ iterations of stage k the entering arcs are of type (g, h) and $d(h) = 2$. The entering arc (g, h) is always chosen using relation (6). To ensure that at every iteration of the algorithm a sole arc will be chosen, the data should be built in such way so that relation (6) is not satisfied by more than one arc (g, h).

To ensure that the algorithm will execute $n(n - 1)/2$ iterations we have to determine a sequence X of $n(n - 1)/2$ entering arcs, so that at stage k we have $n - k$ entering arcs. Let us set

$$X = \{\{(1, 2)\}, \{(2, 2), (2, 3)\}, \ldots, \{(n - 1, 2), (n - 1, 3), \ldots, (n - 1, n)\}\}.$$

We can see that if the above sequence is followed, there is a succession of $n - 1$ stages, where at the ith iteration of stage k the entering arc will be $(n - k, i + 1)$ (see Fig. 2). If we set

$$T_i = \{i^r, (i + 1)^\beta\}$$

(the superscripts r, β indicate a row and column node respectively), we can compute all the parameters associated with the algorithm, if the sequence X is followed (note that T_i is a set of nodes and not a tree data structure). One of the most important variables of the algorithm is the *cut-off* tree T^*. This tree determines the sets F^S, F^D and includes the nodes that are cut-off the current tree when the leaving arc is discarded. From now on, we use notation $T^*_{i,k}$, $F^S_{i,k}$, $F^D_{i,k}$ to denote the cut-off tree T^*, the supply set F^S, and the demand set F^D at iteration i of stage k, $i \leq n-k$, $k = n-1, \ldots, 0$, respectively.

Lemma 1. *If the sequence X of entering arcs is followed then*

$$T^*_{i,k} = \begin{cases} T_i, & i < n-k \\ \left(\bigcup_{j=1}^{n-k-1} T_j\right) \cup \{i^r\}, & i = n-k. \end{cases}$$

Proof. For $i < n-k$ the entering arc is the arc $(n-k, i+1)$. The discarded arc is always of type $(f, i+1)$ and hence $(i+1)^\beta \in T^*$ (Fig. 2a). Additionally, for $i < n-k$ it is $i^r \in T^*$, because at the last iteration of each stage an arc of type $(i, i+1)$ enters the basis. Hence node i^r will always be a leaf of the current tree. For $i = n-k$ (Fig. 2b), an isolated node must obtain children (in order that the algorithm enters the next stage). The entering arc is now the arc $(n-k, n-k+1)$. Now the discarded arc is of type $(n-k, f)$ and T^* will be the subtree rooted on $n-k$, which was created during the preceding $n-k-1$ iterations of stage k. Hence the result follows. □

Lemma 2. *If the sequence X of entering arcs is followed then*

$$F^S_{i,k} = \left(\bigcup_{j=0}^{k} \{(n-k+j)^r\}\right) \cup \left(\bigcup_{j=2}^{i} \{(j-1)^r\}\right)$$

and

$$F^D_{i,k} = \bigcup_{j=i+1}^{n} \{j^\beta\}.$$

Proof. We will use induction. For $k = n-1$ and $i = 1$ the relations hold. Suppose the relations hold for an integer k and an integer i such that $i \leq n-k$. We will prove that the next sets F^S, F^D computed by the algorithm during the next iteration are given by the relations of the Lemma. Note that these sets can be either $F^S_{i+1,k}$, $F^D_{i+1,k}$ or $F^D_{1,k-1}$, $F^D_{1,k-1}$. Thus we distinguish two cases.

Case 1: $i < n-k$

In this case it must be $F^S = F^S_{i+1,k} = F^S_{i,k} \cup T^*_{i,k}$ (the set F^S is expanded). Thus, by Lemma 1 we have that

$$\begin{aligned} F^S_{i,k} \cup T^*_{i,k} &= \left(\bigcup_{j=0}^{k} \{(n-k+j)^r\}\right) \cup \left(\bigcup_{j=2}^{i} \{(j-1)^r\}\right) \cup T_i \\ &= \left(\bigcup_{j=0}^{k} \{(n-k+j)^r\}\right) \cup \left(\bigcup_{j=2}^{i+1} \{(j-1)^r\}\right) \\ &= F^S_{i+1,k} \end{aligned}$$

as

$$\left(\bigcup_{j=2}^{i} \{(j-1)^r\}\right) \cup T_i = \left(\bigcup_{j=2}^{i} \{(j-1)^r\}\right) \cup \{i^r\} = \bigcup_{j=2}^{i+1} \{(j-1)^r\}.$$

Additionally, it must be $F^D = F^D_{i+1,k} = F^D_{i,k} \sim T^*_{i,k}$ (the set F^D is reduced). So

$$F^D_{i,k} \sim T^*_{i,k} = \left(\bigcup_{j=i+1}^{n} \{j^\beta\}\right) \sim T_i = \left(\bigcup_{j=i+1}^{n} \{j^\beta\}\right) \sim (i+1)^\beta$$

$$= \bigcup_{j=i+2}^{n} \{j^\beta\} = F^D_{i+1,k}.$$

Case 2: $i = n - k$

In this case it must be $F^S = F^S_{1,k-1} = F^S_{n-k,k} \sim T^*_{n-k,k}$ (the set F^S is now reduced and the algorithm enters the next stage $k-1$). Hence

$$\begin{aligned}
F^S_{n-k,k} \sim T^*_{n-k,k} &= \left(\bigcup_{j=0}^{k} \{(n-k+j)^r\}\right) \cup \left(\bigcup_{j=2}^{n-k} \{(j-1)^r\}\right) \\
&\sim \left(\left(\bigcup_{j=1}^{n-k-1} T_j\right) \cup \{(n-k)^r\}\right) \\
&= \left(\bigcup_{j=0}^{k} \{(n-k+j)^r\}\right) \cup \left(\bigcup_{j=2}^{n-k} \{(j-1)^r\}\right) \sim \bigcup_{j=1}^{n-k} \{j^r\} \\
&= \left(\bigcup_{j=0}^{k} \{(n-k+j)^r\}\right) \cup \left(\bigcup_{j=1}^{n-k-1} \{j^r\}\right) \sim \bigcup_{j=1}^{n-k} \{j^r\} \\
&= \left(\bigcup_{j=0}^{k} \{(n-k+j)^r\}\right) \sim \{(n-k)^r\} \\
&= \bigcup_{j=0}^{k-1} \{(n-k+1+j)^r\} = F^S_{1,k-1}.
\end{aligned}$$

Additionally, it must be $F^D = F^D_{1,k-1} = F^D_{n-k,k} \cup T^*_{i,k}$ (the set F^D is now expanded). Hence

$$\begin{aligned}
F^D_{n-k,k} \cup T^*_{n-k,k} &= \left(\bigcup_{j=n-k+1}^{n} \{j^\beta\}\right) \cup \left(\bigcup_{j=1}^{n-k-1} T_j\right) \\
&= \left(\bigcup_{j=n-k+1}^{n} \{j^\beta\}\right) \cup \left(\bigcup_{j=1}^{n-k-1} \{(j+1)^\beta\}\right) \\
&= \left(\bigcup_{j=n-k+1}^{n} \{j^\beta\}\right) \cup \left(\bigcup_{j=2}^{n-k} \{j^\beta\}\right) \\
&= \bigcup_{j=2}^{n} \{j^\beta\} = F^D_{1,k-1}.
\end{aligned}$$

Hence the relations hold for all integers $k = n-1, n-2, \ldots, 0$ and $i \le n-k$. □

4. Proof of correctness

According to what has been mentioned above, the algorithm should begin its execution with a matrix s so that the sequence X is achieved. Let us set

$$s = \begin{pmatrix}
0 & s_{12} & M & M & M & \ldots & M \\
0 & s_{22} & s_{23} & M & M & \ldots & M \\
0 & s_{32} & s_{33} & s_{34} & M & \ldots & M \\
0 & s_{42} & s_{43} & s_{44} & s_{45} & \ldots & M \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & s_{(n-1)2} & s_{(n-1)3} & s_{(n-1)4} & s_{(n-1)5} & \ldots & s_{(n-1)n} \\
0 & M & M & M & M & M & M
\end{pmatrix} \tag{8}$$

where M is a very big number. Below, we present two lemmas with their proofs.

Lemma 3. *If a row i of the matrix s is initially sorted in increasing order, then stage $n-i$ will take exactly i iterations to finish.*

Proof. Suppose the algorithm is at the beginning of stage $n-i$ and until this stage the entering arcs have been chosen according to the sequence X. The first iteration of stage $n-i$ begins. By Lemma 2 we get $F^S_{1,n-i} = \{i^r, (i+1)^r, \ldots, n^r\}$ and $F^D_{1,n-i} = \{2^\beta, 3^\beta, \ldots, n^\beta\}$. As from hypothesis $s_{i2} < s_{i3} < s_{i4} < \cdots < s_{i(i+1)}$, arc $(g,h) = (i,2)$ enters the basis. Suppose now the algorithm has executed the first $u-1$ iterations of stage $n-i$ where the remaining arcs $(i,3), (i,4), \ldots, (i,u)$ enter the basis and that after these arcs have entered the basis, the increasing order of row i does not change, i.e., $s_{i(u+1)} < s_{i(u+2)} < s_{i(u+3)} < \cdots < s_{i(i+1)}$. After (i,u) enters the basis, by Lemma 1 we have that $T^*_{u-1,n-i} = T_{u-1} = \{(u-1)^r, u^\beta\}$. Thus only the entries of row $u-1$ and column u are updated, something that preserves the increasing order between the elements $s_{i(u+1)}, s_{i(u+2)}, \ldots, s_{in}$. At the next iteration (iteration u) of stage $n-i$ we will have

$$F^S_{u,n-i} = F^S_{u-1,n-i} \cup \{(u-1)^r\} = \{1^r, 2^r, \ldots, (u-1)^r, i^r, (i+1)^r, \ldots, n^r\}$$

and

$$F^D_{u,n-i} = F^D_{u-1,n-i} \sim \{u^\beta\} = \{(u+1)^\beta, (u+2)^\beta, \ldots, n^\beta\}.$$

As we can easily see, the update of the elements of column u cannot affect the uth iteration of stage $n-i$, as $u^\beta \notin F^D_{u,n-i}$. Additionally the elements of rows $1^r, 2^r, \ldots, (u-1)^r$, although they belong to $F^S_{u,n-i}$, cannot affect the order due to the big number M. Thus we have proved that the next entering arc is the arc $(g,h) = (i,u+1)$ and that the increasing order does not change. By induction, we get that stage $n-i$ will choose the arcs according to the sequence X and therefore take exactly i iterations to finish if row i is sorted in increasing order. □

Lemma 4. *The algorithm will take exactly $n-1$ stages to terminate, executing at the same time the maximum iterations at every single stage k $(n-k)$, if matrix s satisfies the condition $s_{ij} - s_{i(j-1)} > s_{(i-1)j} - s_{(i-1)(j-1)}$, $3 \le j \le i \le n-1$.*

Proof. Let us set

$$0 < s_{12} < s_{22} < s_{23} < s_{32} < s_{33} < s_{34} < \cdots < s_{(n-1)2} < \cdots < s_{(n-1)n}. \tag{9}$$

By Lemma 2 we get

$$F^S_{1,n-1} = \{1^r, 2^r, \ldots, n^r\} \wedge F^D_{1,n-1} = \{2^\beta, 3^\beta, \ldots, n^\beta\}.$$

The first entering arc is the arc (1, 2). By Lemma 1 we get $T^*_{1,n-1} = \{1^r\}$. According to the update method used by the algorithm, s_{12} is subtracted from row 1 of matrix s. Thus $s_{12} = 0$. Then arc (2, 2) enters the basis. Because of Lemma 3 and relation (9) the next entering arc is arc (2, 3). This is the second iteration of stage $n-2$. During the first iteration of stage $n-2$, where $T^*_{1,n-2} = \{1^r, 2^\beta\}$, the following two update operations took place:

$$s_{.2} = s_{.2} - s_{22}e \wedge s_{1.} = s_{1.} + s_{22}e^T.$$

During the second iteration of stage $n-2$, where $T^*_{2,n-2} = \{1^r, 2^\beta\} \cup \{2^r\}$, the following three update methods took place:

$$s_{.2} = s_{.2} + s_{23}e \wedge s_{1.} = s_{1.} - s_{23}e^T \wedge s_{2.} = s_{2.} - s_{23}e^T.$$

Thus the updated matrix s after stages $n-1, n-2$ have elapsed will be

$$s = \begin{pmatrix} s_{22} - s_{12} - s_{23} & 0 & M & M & M & \ldots & M \\ -s_{23} & 0 & 0 & M & M & \ldots & M \\ 0 & s_{32} - s_{22} + s_{23} & s_{33} & s_{34} & M & \ldots & M \\ 0 & s_{42} - s_{22} + s_{23} & s_{43} & s_{44} & s_{45} & \ldots & M \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & s_{(n-1)2} - s_{22} + s_{23} & s_{(n-1)3} & s_{(n-1)4} & s_{(n-1)5} & \ldots & s_{(n-1)n} \\ 0 & M & M & M & M & M & M \end{pmatrix}. \tag{10}$$

It is obvious that arc (3,2) can only be chosen if and only if

$$s_{32} - s_{22} + s_{23} < s_{33} \Longleftrightarrow s_{33} - s_{32} > s_{23} - s_{22}.$$

If the above relation holds, it is easy to see by Lemma 3 and relation (9) that stage $n-3$ will be executed in full. After stage $n-3$ has elapsed, the reader can verify that the updated matrix s will be

$$s = \begin{pmatrix} -s_{12} - s_{34} & 0 & M & M & M & \dots & M \\ (s_{33} - s_{23} - s_{34}) & (-s_{32} + s_{22} - s_{23} + s_{33}) & 0 & M & M & \dots & M \\ -s_{34} & 0 & 0 & 0 & M & \dots & M \\ 0 & s_{42} - s_{32} + s_{34} & s_{43} - s_{33} + s_{34} & s_{44} & s_{45} & \dots & M \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & s_{(n-1)2} - s_{32} + s_{34} & (s_{(n-1)3} - s_{33} + s_{34}) & s_{(n-1)4} & s_{(n-1)5} & \dots & s_{(n-1)n} \\ 0 & M & M & M & M & M & M \end{pmatrix}. \tag{11}$$

Thus, two relations must now hold:

$$s_{42} - s_{32} + s_{34} < s_{43} - s_{33} + s_{34} \Longleftrightarrow s_{43} - s_{42} > s_{34} - s_{33}$$

and

$$s_{43} - s_{33} + s_{34} < s_{44} \Longleftrightarrow s_{44} - s_{43} > s_{34} - s_{33}.$$

Applying the same method, it is easy to see that the algorithm will pass from one stage to the next "in full" if the following $i-2$ inequalities hold for every row i of the initial matrix s:

$$s_{ij} - s_{i(j-1)} > s_{(i-1)j} - s_{(i-1)(j-1)},\ 3 \le j \le i \le n-1. \quad \square \tag{12}$$

In the next theorem we present the main result of our research work.

Theorem 5. *Let c be a matrix of non-negative integers, where*

$$c_{ij} = \begin{cases} i \times j, & i = 1, \dots, n-1 \wedge j = 2, \dots, i+1 \\ M, & i = 1, \dots, n-1 \wedge j = i+2, \dots, n \\ M, & i = n \wedge j = 2, \dots, n \\ 0, & i = 1, \dots, n \wedge j = 1 \end{cases} \tag{13}$$

where $M = n(n-1) + x$, $x \in \mathbb{R}$. Matrix c belongs to the class of worst case data matrices for the EPSA applied to the Assignment Problem.

Proof. Firstly, we must compute the reduced cost matrix s. It easy to see that $s = c$. From Lemma 4 we know that the algorithm will execute the maximum number of iterations if relation (12) holds. If we substitute relation (13) to relation (12) we have

$$\begin{aligned} &ij - i(j-1) > (i-1)j - (i-1)(j-1) \Longleftrightarrow \\ &ij - ij + i > ij - j - ij + i + j - 1 \Longleftrightarrow \\ &0 > -1 \end{aligned} \tag{14}$$

which holds for all integers i, j. $\square$

5. Computational results

In this section we describe our numerical tests, which demonstrate the behavior of exterior point algorithms efficiency on the proposed worst case examples. All test runs were carried out on a 1.7 GHz Pentium M Intel processor with 512 Mb memory. All codes are compiled using Matlab 7.0.4 with default options.

Three variations of exterior point simplex algorithms are compared in our computational study. These algorithms are:

Table 1
Results on worst case instances for Exterior Point Algorithms

n	$\frac{n(n-1)}{2}$	Bal		DF		SF		Obj. value
		niter	cputime	niter	cputime	niter	cputime	
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
25	300	253	0.1125	300	0.1469	324	0.1656	5 200
50	1 225	1 128	0.9625	1 225	1.1437	1 275	1.2875	41 650
75	2 775	2 628	3.5172	2 775	4.0078	2 850	4.5016	140 600
100	4 950	4 753	8.9766	4 950	9.9875	5 050	11.2125	333 300
125	7 750	7 503	18.8500	7 750	20.5609	7 875	23.0922	651 000
150	11 175	10 878	34.9109	11 175	37.5609	11 325	42.1797	1 124 950
175	15 225	14 878	59.1234	15 225	62.9016	15 399	70.7359	1 786 400
200	19 900	19 503	93.7906	19 900	98.9516	20 099	111.4422	2 666 600
225	25 200	24 753	142.0406	25 200	148.4141	25 425	167.3969	3 796 800
250	31 125	30 628	205.7422	31 125	213.1812	31 374	241.5391	5 208 250
275	37 675	37 128	293.0984	37 675	302.0031	37 949	343.3062	6 932 200
300	44 850	44 253	401.1953	44 850	410.1531	45 149	467.2578	8 999 900
325	52 650	52 003	534.7187	52 650	544.1766	52 975	619.9422	11 442 600
350	61 075	60 378	700.2594	61 075	708.9906	61 425	808.3891	14 291 550

- The Dual Forest algorithm presented in Section 2.2. We denote this algorithm by DF.
- The exterior point simplex algorithm starting with Balinksi's feasible tree [10]. We denote this algorithm by Bal.
- The exterior point simplex algorithm starting with a simple forest [9]. We denote this algorithm by SF.

We solved 14 different classes ($n = 25, 50, \ldots, 350$) of worst case examples. In Table 1 we present our computational results. The reported cpu times were measured in seconds with Matlab's built-in function cputime. Column (1) indicates the size of the worst case assignment problem, column (2) indicates the maximum number of iterations according to the theoretical worst case behavior of the dual forest algorithm (DF) while column (5) the executed number of iterations for the same algorithm. For each class, columns (2) and (5) have the same value (number of iterations), a result which is expected, since it has been proved in the previous section. In columns (3) and (7) we report the number of iterations while in columns (4) and (8) the cpu time for the algorithms Bal and SF respectively. Finally, in column (9) we report the optimal objective value for each problem class. All algorithms terminated with correct optimal objective values. This computational study verifies that the worst case examples generated using our scheme (see, Theorem 5) are hard not only for the Dual Forest EPSA but also for the other two exterior point algorithms.

6. Conclusions

The design of experiments appears to be an integral and vital part of the whole scientific process. In this paper we presented a generation scheme of worst case examples of the EPSA for the APs. By considering a set of mathematical properties we proposed a general framework for generating worst case examples. These examples play an important role in algorithm analysis. Using this class of problems as a confirmation tool, we can verify the relation between theoretical results and practical performance.

Acknowledgements

The authors wish to thank the anonymous reviewers for some very crucial comments and suggestions that helped them to improve the quality of the paper.

References

[1] M. Dell' Amico, P. Toth, Algorithms and codes for dense assignment problems: The state of the art, Discrete Appl. Math. 100 (2000) 17–48.
[2] M. Akgül, The linear assignment problem, in: M. Akgul, H.W. Hamacher, S. Tfekci (Eds.), Combinatorial Optimization NATO ASI series F, vol. 82, 1992, pp. 85–122.

[3] U. Derigs, The shortest augmenting path method for solving assignment problems-motivation and computational experience, in: C.L. Monma (Ed.), Algorithms and Software for Optimization — Part I, Ann. Oper. Res. 4 (1985), 57–102.
[4] S. Martello, P. Toth, Linear assignment problems, in: S. Martello, G. Laporte, M. Minoux, C. Ribeiro (Eds.), Surveys in Combinatorial Optimization, Ann. Discrete Math. 31 (1987), 259–282.
[5] M.L. Balinski, The hirsch conjecture for dual transportation polyhedra, Math. Oper. Res. 9 (1984) 629–633.
[6] M.L. Balinski, Signature method for the assignment problem, Oper. Res. 33 (1985) 527–536.
[7] M.L. Balinski, A competitive (dual) simplex method for the assignment problem, Math. Program. 34 (1986) 125–141.
[8] P. Kleinschmidt, C.W. Lee, H. Schannath, Transportation problems which can be solved by the use of hirsch-paths for the dual problems, Math. Program. 37 (1987) 153–168.
[9] H. Achatz, K. Paparrizos, N. Samaras, K. Tsiplidis, A forest exterior point algorithm for the assignment problems, in: P.M. Pardalos, A. Migdalas, A. Buckard (Eds.), Combinatorial and Global Optimization, Word Scientific Publishing Co., 2002, pp. 1–10.
[10] K. Paparrizos, An infeasible (exterior point) simplex algorithm for assignment problems, Math. Program. 51 (1991) 45–54.
[11] H. Achatz, P. Kleinschmidt, K. Paparrizos, A dual forest algorithm for the assignment problem, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 4 (1990) 1–10.