Vol. 00, No. 00, August 2008, 1–17

On the initialization methods of an exterior point algorithm for the assignment problem

C. Papamanthou†, K. Paparrizos‡, N. Samaras§ ‡ and A. Sifaleras‡

†Department of Computer Science, Brown University, Providence RI, U.S.A.
‡ Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece

(*v3.2 released December 2006*)

In this paper we present a theoretical investigation and an extensive computational study of Exterior Point Simplex Algorithm (EPSA) initialization methods for the Assignment Problem (AP). We describe the exterior point algorithm using three different initialization methods. Effective implementations are explored for each initialization method. Then we perform an experimental evaluation on a large set of benchmark problems from the TSPLib95 and OR Library collections. The results obtained demonstrate the advantages of the three initialization methods. Finally, we give a theoretical justification of the initialization methods efficiency. We explain theoretically the computational ranking for these methods.

Keywords: Combinatorial Optimization; Assignment Problem; Exterior Point Algorithm; Initialization Methods; Computational Evaluation.

AMS Subject Classifications: 90C27; 05C85; 90B10; 65K05; 91A90

1. Introduction

The Assignment Problem (AP) is one of the most well studied problems in mathematical programming. The AP has various applications in the real world. It could be used to model the assignment of employees to tasks, or machines to productions jobs, but its uses are more widespread. For example, it could be used in computer networking or in assigning aircrafts to trips. The AP is a Hitchcock transportation problem. The only difference is that the supply (demand) at every supply (demand) node is equal to one.

A large number of algorithms has been developed for the AP. 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. Primal-dual algorithms work with a pair of an infeasible primal solution and a feasible dual solution which satisfy the complementarity slackness conditions. The most well-known algorithms of this category are the Hungarian method [1] and the auction algorithm [2]. The Hungarian method is the first non-simplex algorithm for the AP.

The simplex type algorithms are modifications of the classical network simplex algorithm. According to the nature of the initial basic solution, simplex type algorithms can be further divided into two subcategories; primal and dual simplex

§Corresponding author Email: samaras@uom.gr

International Journal of Computer Mathematics
ISSN: 0020-7160 print/ISSN 1029-0265 online
http://www.tandf.co.uk/journals
DOI: 10.1080/0020716YYxxxxxxxx

type algorithms. Dual simplex type algorithms for the AP work with a spanning tree which defines a dual feasible basic solution. At every iteration a pivoting operation is performed on an arc for which the corresponding primal constraint is violated. The use of different pivoting rules for the selection of the leaving and entering arcs resulted in different versions of dual simplex algorithms. Balinski [3] introduced a competitive dual simplex algorithm for the AP with $O(n^2)$ pivot and $O(n^3)$ time complexity. Goldfarb [4] developed a different signature method which solves a sequence of smaller problems of the given AP. An algorithm which works with strongly dual feasible trees has been developed by Akgul [5]. Its worst case complexity is $O(n^3)$ or $O(nm+n^2logn)$ depending of data structure used. A similar algorithm to Akgul's was proposed by Paparrizos [6].

A non-dual signature method for the AP has been developed by Paparrizos [7]. This algorithm visits only strong trees which are obtained from strongly feasible trees by dropping the feasibility requirement. Its worst case complexity is $O(n^4)$. An efficient implementation of Paparrizos's algorithm was given by Akgul and Ekin [8]. The improvement is that this algorithm updates a forest rather than a tree. The worst case complexity decreased to $O(n^3)$ using elementary data structures. Using Fibonacci Heaps for sparse APs it has $O(n^2logn + nm)$ complexity. Later, Paparrizos [9] developed a new class of simplex type algorithms for the AP with $O(n^3)$ complexity. This class is called Exterior Point Simplex Algorithms (EPSA). An experimental study to compare the classical simplex algorithm and the exterior point algorithm for the transportation problem can be found in [10]. Totally, four algorithms are compared on uniformly randomly generated test problems. The results are very encouraging for the dual forest exterior point algorithm.

A complete survey of computationally attractive algorithms for the classical AP can be found in [11]. Surveys on methods and algorithms which solve the AP have been presented by Martello and Toth [12] and Derigs [13]. Several papers and research work exist which compare algorithms for the AP [14–16]. It is well known, that using an efficient starting solution is essential. Very often differences in running time are not due to different algorithmic approaches (such as primal, dual, primal-dual, etc.), but are due to the various starting procedures which are used.

The aim of this research work is twofold. First, to perform an extended computational study between three different initialization methods for the EPSA. Second, and more importantly, to analyze the three different approaches theoretically and to give some additional algorithmic insight into why the winners won. Toward these aims, we make the following contributions:

- We describe the exterior point simplex algorithm using three different initialization methods. These methods are the exterior point simplex algorithm starting with
  (i) Balinski's feasible tree [9]
  (ii) A simple forest [17] that is neither primal nor dual feasible and
  (iii) A feasible forest [18]

  The exterior point simplex algorithm using the last initialization method solves an AP in at most $\frac{n(n-1)}{2}$ iterations and in at most $O(n^3)$ time. A detailed visual representation of the above three initializations is described in [19]
- We perform an extensive computational study on various dense benchmark APs from the TSPLib95 and OR Library collections. This study consists of two parts. In the first part, we report results showing the advantages of the initialization methods for the APs testbed. In the second part we report the computation of the column level of a solution for each one of the benchmark APs. The column level declares the distance of an initial solution from the optimal solution. Roughly

speaking, column level is one of the main factors that determines the quality of the initial solution.

- Finally, we give a theoretical justification of the three diferrent initialization methods efficiency. The computational efficiency of all Simplex type algorithms depends on
  - (i) The distance between the initial solution and the optimal solution and
  - (ii) The structure of the initial solution.

  We explain theoretically the computational ranking for the three competitors.

The paper is organized as follows. Following this introduction, in Section 2 we give the necessary mathematical background and we briefly present the exterior point simplex algorithm. In Section 3 we present the three different initialization methods. We also demonstrate these methods using an illustrative example. In Section 4 we present experimental results on benchmark APs that demonstrate the effectiveness of the three initialization methods. A theoretical explanation of the three initialization methods efficiency is presented in Section 5. Finally, in Section 6 we conclude and discuss future work.

2. Exterior Point Algorithm description

It is well known that an AP can be represented by bipartite graph $G(S, D, E) = G(N, E)$, which consists of two discrete sets of nodes S, D, $(N = S \cup D, S \cap D = \emptyset)$ 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 column (supply) nodes, whereas nodes $j \in D$ are called row (demand) nodes. In our Figures we draw column nodes as circles and row nodes as squares. The mathematical formulation of the linear AP with a square $(n \times n)$ cost matrix C is the following:

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

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

$$\sum_{j=1}^{n} x_{ij} = 1, \; i = 1, 2, \ldots, n \tag{3}$$

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

Problem (LAP) can be formulated by means of the integer linear programming problem defined by Equations (1), (2), (3), and replacing (4), with the constraints

$$x_{ij} \in Z, \quad 0 \leq x_{ij} \leq 1, 1 \leq i, j \leq n$$

Obviously, $x_{ij} = 1$ or 0. In particular, $x_{ij} = 1$ if column j is assigned to row i. The associated dual problem to (LAP) is:

$$(DLAP) \qquad \max \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j$$

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

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

Given a pair of feasible solutions x and (u, v) for the problems (LAP) and (DLAP) respectively, the complementary slackness condition is stated as

$$x_{ij}s_{ij} = 0, (i, j = 1, 2, \dots, n), \quad where \quad s_{ij} = c_{ij} - u_i - v_j \tag{5}$$

By s_{ij}, we denote the reduced cost variable corresponding to the variable - arc (i, j). In our implementation, all the reduced cost variables are stored in a square (nxn) matrix. From now on we will assume that this square (nxn) matrix has full dense, since all the benchmark instances in our computational study of Section 4, have full dense cost matrics. Furthermore, by $s_{i.}$ we denote a row vector corresponding to the i row of that square (nxn) matrix. For example, by $s_{4.}$ we denote a row vector containing all the reduced cost variables which corresponds to arcs leaving from the 4^{th} supply node. Therefore, $s_{4.} = (s_{41}, s_{42}, s_{43}, ..., s_{4n})$. Similarly, by $s_{.i}$ we denote a column vector corresponding to the i column of that square (nxn) matrix. For example, by $s_{.4}$ we denote a column vector containing all the reduced cost variables which corresponds to arcs coming to the 4^{th} demand node. Therefore, $s_{.4}^T = (s_{14}, s_{24}, s_{34}, ..., s_{n4})$. Moreover, by e we denote a unit column vector of n elements, while the transpose of e will be denoted by e^T. For example, the unit row vector is $e^T = (1, 1, ..., 1)$.

Roughly speaking Exterior Point Simplex Algorithms, for the AP, are initialized with a feasible tree or forest. The main difference between them and the dual simplex type algorithms is that exterior point algorithms don't maintain dual feasibility on every one iteration. Dual feasibility is destroyed and restored again at the optimal solution. The main idea of EPSA is as follows. At each iteration a solution T is computed. T is a directed rooted tree. It represents the current assignments for each individual solution. Every arc (i, j) of the tree is directed from a column node to a row node and represents the temporary assignment of facility j to a user i. The set of nodes are partitioned into two subsets F and $T \setminus F$. The algorithm stops when $F = \emptyset$. Both sets T and F depend on the initalization method, which means that different initialization methods produce different starting trees and forests. Following in Section 3, there is analytical description of the initialization of those two sets. Furthermore, there is an explanatory example in Section 3.4 with an accompanying Figure 4.

As mentioned in the introduction, EPSA is a simplex type algorithm. This means that at every iteration an arc enters the basis (entering variable) and an arc leaves the basis (leaving variable). In our case the basis is a tree which contains all the variables that are arcs of the current solution T. Specifically, 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 $s_{gh} = \min\{s_{ij} : i \in F, j \in T \setminus F\}$. Then the leaving arc (k, l) is chosen. The EPSA's versions using three different initialization methods considered in this paper, differ among each other in the way their starting tree structures are initialized, the way subset F is constructed and the way the leaving arc (k, l) is chosen. EPSA also uses a special data structure - tree, which from now on will be denoted by T^*. More specifically, if the leaving arc was discarded before the selection of the entering arc, then the current solution (T tree), would have been divided into two sub-trees. We denote by T^*, the sub-tree which wouldn't contain the root node. The T^* tree is very important in the implementation, because only the reduced costs of arcs with one of its nodes belonging to the tree T^* and the other to the subtree $T \setminus T^*$ are updated. Finally, the T^* tree determines the sets $F, T \setminus F$. In the pseudocode below we describe the main steps of EPSA.

Algorithm 1

Require: $G = (N, E), c, T$

procedure EPSA(G, T)
    Start with a special solution T. Determine the subsets $F, T \setminus F$ and compute s_{ij} according to the initialization method.
    while $F \neq \emptyset$ **do**
        $\delta = s_{gh} = \min \{s_{ij} : i \in F \wedge j \in T \setminus F\}$
        choose the leaving arc (k, l) according the special rules of each initialization method.
        Update tree using $T' = T \cup (g, h) \setminus (k, l)$
        if $h \in T^*$ **then**
            $q = -\delta$
        else
            $q = \delta$
        end if
        for $i \leftarrow 1, n$ **do**
            if row node $i \in T^*$ **then**
                set $s_{i.} = s_{i.}(T) - qe^T$
            end if
            if column node $j \in T^*$ **then**
                set $s_{.j} = s_{.j}(T) + qe$
            end if
        end for
        set $T = T'$
    end while
end procedure

3. Initialization methods for EPSA

3.1. *The Balinski tree*

Balinski tree is dual feasible. Its root is row node 1 and all column nodes lie below the root at depth 1. The remaining $(n-1)$ row nodes are connected to the column nodes of the tree and thus lie at depth 2. Let T denote the Balinski tree. The standard arcs of the tree are the arcs of type $(1, j)$, where j is a column node. Initially, we set $u_1(T) = 0$ and $v_j(T) = c_{ij},\ j = 1, 2, \ldots, n$. Hence, for each arc $(1, j)$ we compute the reduced costs using the relation $s_{1j}(T) = c_{1j} - u_1(T) - v_j(T) = 0$. Given a row index i, the column index associated with the minimum $c_{ij} - v_j$ value of row i is

$$j(i) = arg \min \{c_{ij} - v_j,\ j = 1, 2, \ldots, n\}$$

The remaining dual variables $u_i(T)$ can be computed by setting $u_i(T) = \min\{c_{ij} - v_j(T),\ j = 1, 2, \ldots, n\},\ i = 1, 2, \ldots, n$. Let now $u_i(T) = c_{ij(i)} - v_{j(i)},\ i = 2, 3, \ldots, n$. Then $(i, j(i)) \in T$. At this point all arcs belonging to the Balinski tree have been determined. Now we can easily compute the reduced costs of the non-basic variables using relation (5). The decision variables x_{ij} corresponding to basic arcs (i, j) of the Balinski tree are computed easily by setting $x_{ij} = 1,\ i = 2, 3, \ldots, n$ and $x_{ij} = 1$, if j is a leaf of the tree. For the remaining arcs of type $(1, j)$, where column node j has at least one child, we set $x_{1j} = 1 - m_j \leq 0$. We use variable m_j to store the number of children of column node j. Finally, the forest F is the set of trees $\{T_j : x_{1j} < 0\}$, where T_j is the subtree rooted on column node j. Obviously, $T \setminus F$ is a subtree of T. A general Balinski tree for an AP of size n can be seen at Figure 1.

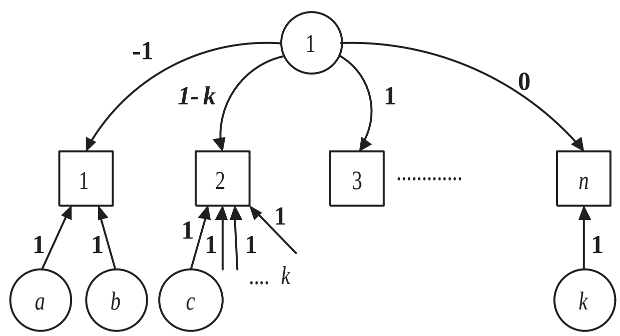

Figure 1. The general Balinski tree for an $(n \times n)$ Assignment Problem.

3.2. *The Simple Start forest*

Let the simple start forest be denoted by Q. Forest Q consists of the $2n$ isolated nodes. The set F consists of all the row-nodes, while the set $T \setminus F$ of all the column nodes. Thus, $F = \{i : i \in S\}$ and $T \setminus F = \{j : j \in D\}$. Additionally, we set

$$u_i(Q) = v_j(Q) = 0,\ i \in S,\ j \in D \tag{6}$$

Thus, by replacing conditions 6 in relation (5) we take

$$s_{ij}(Q) = c_{ij}$$

A general simple start forest for an AP of size n can be found at Figure 2.

Figure 2. The general simple start forest for an $(n \times n)$ Assignment Problem.

To adjust the simple start forest to the needs of EPSA we need to transform the forest Q to a tree T. This transformation can easily be done by inserting an artificial node 0, which is the root of tree T and adding $2n$ artificial arcs. For each row node i an artificial arc $(i, 0)$ with unit cost $c_{i0} = 0$ is introduced. Similarly, for each column node j an artificial arc $(0, j)$ with unit cost $c_{0j} = 0$ is introduced. All the basic decision variables of type $x_{i0},\ i = 1, 2, \ldots, n$ and $x_{0j},\ j = 1, 2, \ldots, n$ are initially set equal to 1. EPSA updates the sets F and $T \setminus F$ using the relations $F = \{T_i\ :\ i \in S,\ (i, 0) \in T\}$ and $T \setminus F = \{T_j\ :\ j \in D\}$.

3.3. *The dual feasible forest*

Let Q denote this forest, which consists of n subtrees T_j. Let $u_i(Q),\ v_j(Q),\ i, j = 1, 2, \ldots, n$, denote the dual variables that correspond to the column and the row nodes of the forest respectively. Initially we set

$$v_j(Q) = 0,\ \forall j \in D$$

Next, we compute the dual variables of the column nodes $u_i(Q)$ by setting

$$u_i(Q) = \min \{c_{ij}\ :\ j = 1, 2, \ldots, n\}, \forall i \in S$$

For the same reason referred to at the simple start method, we need to transform the forest Q to a tree T. This can be done by inserting an artificial node 0 (root of the tree T) and n artificial arcs. For each column node j such that $T_j \in F$, (j is the root of subtree T_j), an artificial arc $(j, 0)$ with unit cost $c_{j0} = 0$ is introduced. Similarly, for each column node $j \notin F$ an artificial arc $(0, j)$ with unit cost $c_{0j} = 0$ is introduced. Finally, it is set $x_{0j}(T) = |d(j) - 2|$ and $x_{j0}(T) = |d(j) - 2|$, where $d(j)$ is the degree of node j after the insertion of the artificial arcs.

Let p be the column node such that $u_i(Q) = c_{ip}$. Then, the arc (i, p) is a basic arc of the forest Q. For each arc (i, p) we set $x_{ip} = 1$. The reduced costs s_{ij} can be computed using Relation 5. At this point note that $s_{ij}(Q) \geq 0$, which means that forest Q is dual feasible. The initial set F is $F = \{T_j\ :\ d(j) < 2\}$. A general dual feasible forest for an AP of size n is illustrated in Figure 3.

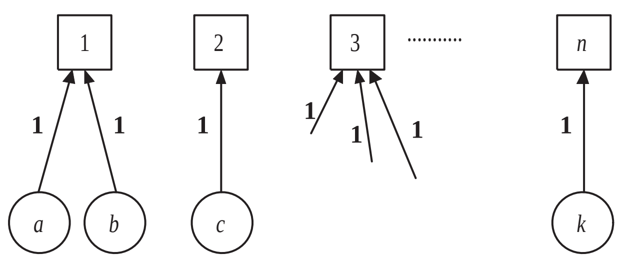

Figure 3. The general AKP forest for an $(n \times n)$ Assignment Problem.

3.4. *An illustrative example*

The cost matrix $C = \{c_{ij}\},\ \forall (i, j) \in A$ of order n is the input data in an AP. Suppose, that we want to solve the AP having the following cost matrix

$$C = \begin{bmatrix} -7 & 7 & 8 & 1 \\ 0 & -1 & 2 & 9 \\ 3 & 0 & 9 & 1 \\ 1 & 12 & 4 & 5 \end{bmatrix}$$

If we apply the three initialization methods mentioned above in the example we take, in Figure 4, the following feasible solutions

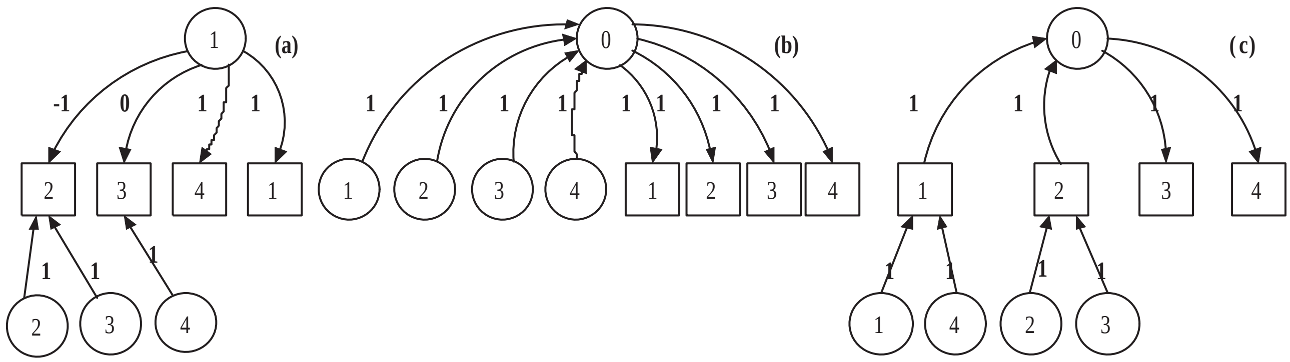

Figure 4. Starting solutions: (a) Balinski, (b) Simple Start, (c) Dual Feasible Forest.

4. Computational Experiments

In this section we present our numerical results and we briefly discuss some important implementation characteristics of the three initialization methods. In order to test the computational behavior of the three different initialization methods we have used benchmark instances from TSPLib95 [20] and OR-Library [21]. The TSPLib95 and OR-Library are well known suites containing many hard to solve optimization problems. All instances have full dense cost matrices. The cost matrix $C = \{c_{ij}\} \forall\ (i, j) \in A$ of order n is the input data in an AP. The order of the cost matrix varies up to 1400 rows and columns. Our numerical experiments were performed on a PC with 2.4 GHz P4 processor, RAM 512 Mb and with Windows XP Pro operating system. The three initialization methods have been programmed in MATLAB 7.0.1 in exactly the same way. This means that the codes used have been written following the same programming techniques adjusted every time to the special characteristics of each initialization method. For each collection, the results are summarized with a table and a figure. The reported CPU times were computed with the built-in function cputime. The given times are net times and do not include times for the input.

Moreover, in order to compute the nodes of the T^* tree, we have used a function that returns the vector of the preorder traversal of a tree of root x. This preorder traversal can be implemented non-recursively or recursively. During the first phase of the EPSA implementation we used the non-recursive version of the preorder traversal. We observe that nearly 50% of the CPU time was absorbed by this function, increasing in this way the overall execution time of EPSA. For this reason, we implemented preorder traversal recursively. In the rest of the paper the first initialization method is denoted by (Bal), the second by (SS) and the third by (AKP).

4.1. *Benchmark APs (TSPLib 95)*

In order to gain a deeper insight into the practical behaviour of the three initialization methods, we tested them on some benchmark instances taken from TSPLib 95 [20]. TSPLib 95 is a library of sample instances of the Travelling Salesman Problem (TSP) from various sources and of various types. In our study we have solved instances that belong to two classes. These classes are symmetric ($c_{ij} = c_{ji},\ i,j = 1,2,\ldots,n,\ i \neq j$) and assymetric ($c_{ij} \neq c_{ji},\ i,j = 1,2,\ldots,n,\ i \neq j$) TSPs. Given a set of nodes and distances for each pair of nodes, find a roundtrip of minimal total length visiting each node exactly once. Roughly speaking, the AP defines a lower bound for TSP. For this reason it is important to test the efficiency of the initialization methods on these benchmark instances. All the diagonal elements of the TSPs are equal to zero. In order to solve these instances as APs using EPSA, we assigned to these entries, $c_{ii},\ i = 1,2,\ldots,n$ a large positive value M. This positive value is equal to 10^8.

In Table 1 we present our computational results on symmetric TSPs by alphabetical order. The total number of symmetric TSPs solved is 71. These instances are all very different from each other, especially in the structure of the cost matrix. The entries of the cost matrix are computed using various distance functions. These functions are Euclidean distance, pseudo-Euclidean distance and geographical distance. For more details see [20]. Also, the size n of the instances ranges from 14 to 1400. The first two columns of Table 1 contain the name and the size of the TSP. In column 3 the optimal value of the objective function for each one instance is displayed. The remaining columns (columns 4 to 9) show the number of iterations and CPU time in seconds for each one of the three initialization methods. The last row of Table 1 defines the average number of iterations and the average CPU time over all test instances.

Before analyzing the results collected in Table 1 we would like to warn the reader that the computational results depend on many factors. For example, the choice of instances, the structure of the cost matrix, the choice of programming language, the choice of computing environment, all these factors influence the relative performance of the three initialization methods. All TSPs were solved within the time limit. The average number of iterations for the initialization methods Bal, SS and AKP is 1309.155, 741.197 and 416.380 iterations respectively. The corresponding average CPU times (in seconds) are 54.695, 34.306 and 18.778. Table 2 contains the normalized ratios taken from Table 1. As we can see from Table 2, AKP is about 3.176 times faster than Bal in terms of number of iterations and about 3.238 times faster in terms of CPU time over all symmetric TSPs. From the same Table we observe that AKP is also faster than SS over all instances. Particularly, AKP is about 1.940 times faster than SS in terms of number of iterations and about 2.105 times faster in terms of CPU time.

In order to show more clearly the superiority of the initialization method AKP over the other methods we plot the number of iterations and the CPU time of the seven largest symmetric TSPs (Figures 5 and 6).

Table 1. TSPLib95: Symmetric instances, (time in seconds)

name	n	zvalue	Bal		SS		AKP	
			niter	cpu	niter	cpu	niter	cpu
a280	280	2423	1252	8.200	603	4.245	358	2.344
ali535	535	155,915	1870	36.573	1548	31.975	983	18.580
att48	48	8428	127	0.084	94	0.059	47	0.028
att532	532	22,783	2343	46.648	1198	25.523	623	12.578
berlin52	52	6287	134	0.092	124	0.084	62	0.039
bier127	127	95,802	334	0.633	395	0.786	255	0.456
brazil58	58	16,565	154	0.122	127	0.100	62	0.044
burma14	14	2866	31	0.013	29	0.011	16	0.003
ch130	130	4381	384	0.745	222	0.486	90	0.192
ch150	150	5558	551	1.341	299	0.780	150	0.383
d198	198	10,607	654	2.553	485	1.900	288	1.050
d493	493	30,286	2542	43.023	1468	26.555	909	15.227
d657	657	40,561	2532	73.767	1660	51.972	983	27.995
d1291	1291	40,698	4742	500.969	3114	357.234	1859	171.281
eil51	51	376	151	0.098	101	0.067	51	0.031
eil76	76	484	265	0.269	177	0.186	101	0.097
eil101	101	571	345	0.477	238	0.363	134	0.186
fl417	417	7422	1224	16.019	842	11.811	511	6.503
fl1400	1400	11,988	5884	751.659	3962	522.700	3349	422.694
gil262	262	1922	866	5.153	571	3.545	303	1.777
gr96	96	46,319	342	0.472	211	0.298	119	0.156
gr137	137	57,224	557	1.205	317	0.716	178	0.367
gr202	202	34,262	834	3.070	694	2.686	393	1.394
gr229	229	108,973	911	4.038	467	2.308	231	1.089
gr431	431	142,117	2164	29.759	1156	16.595	671	8.936
gr666	666	252,944	3469	98.817	1929	59.417	1210	34.514
kroA100	100	17,087	244	0.331	197	0.295	93	0.131
kroA150	150	21,515	448	1.044	296	0.772	148	0.378
kroA200	200	23,096	684	2.495	404	1.639	197	0.763
kroB100	100	16,791	275	0.391	193	0.288	87	0.128
kroB150	150	20,482	419	1.000	317	0.809	166	0.395
kroB200	200	23,409	709	2.563	368	1.495	179	0.705
kroC100	100	16,738	309	0.433	215	0.320	105	0.142
kroD100	100	16,540	264	0.375	210	0.313	106	0.148
kroE100	100	16,685	283	0.403	182	0.275	80	0.113
lin105	105	8956	284	0.445	212	0.336	92	0.142
lin318	318	27,289	978	7.944	651	5.747	288	2.445
p654	654	23,509	2692	78.889	1576	50.752	825	22.217
pcb442	442	46,830	2304	32.813	1175	17.948	698	9.688
pr76	76	77,119	195	0.194	153	0.158	80	0.078
pr107	107	24,207	275	0.436	181	0.305	92	0.150
pr124	124	38,925	330	0.636	188	0.395	69	0.139
pr136	136	85,552	459	1.019	280	0.648	168	0.363
pr144	144	20,008	354	0.805	208	0.536	68	0.161
pr152	152	43,044	402	1.039	308	0.839	152	0.414
pr226	226	49,937	607	2.853	440	2.153	206	0.972
pr264	264	33,026	888	5.045	493	3.138	224	1.317
pr299	299	39,881	1,292	9.406	540	4.413	248	1.914
pr439	439	76,887	1,471	20.841	842	12.933	415	6.036
pr1002	1002	214,013	4546	294.227	2221	157.386	1207	80.491
rat99	99	1089	403	0.584	217	0.320	117	0.166
rat195	195	2095	723	2.716	382	1.480	193	0.717
rat575	575	6006	2,932	66.128	1,205	29.525	648	15.386
rat783	783	7443	3,808	152.239	1,543	71.867	758	32.819
rd100	100	6559	305	0.453	211	0.317	116	0.164
rd400	400	12,360	1545	17.736	783	10.314	386	4.914
rl1304	1304	189,885	4406	477.436	2518	313.114	1194	130.938
si175	175	20,243	477	1.505	377	1.239	233	0.703
si535	535	45,204	1164	22.261	947	20.731	490	10.005
si1032	1032	90,698	2319	152.338	1626	121.120	680	43.791
st70	70	519	200	0.194	149	0.147	75	0.077
swiss42	42	1009	101	0.064	72	0.045	31	0.016
ts225	225	115,605	1271	6.266	457	2.241	336	1.430
tsp225	225	3418	883	4.039	431	2.083	219	1.023
u159	159	34,649	567	1.484	296	0.856	146	0.405
u574	574	29,110	2750	61.617	1195	29.525	609	14.386
u724	724	35,792	4201	142.858	1478	55.714	733	26.756
u1060	1060	183,314	4994	354.372	2550	201.409	1500	106.028
ulysses16	16	5458	28	0.014	32	0.011	18	0.005
ulysses22	22	5767	43	0.025	48	0.014	26	0.008
vm1084	1084	191,354	4456	327.564	2227	187.350	1126	86.145
Average			**1309.155**	**54.695**	**741.197**	**34.306**	**416.380**	**18.778**

Table 3 demonstrates the performance of the three initialization methods on asymmetric TSPs. The asymmetric TSPs have at most 171 rows and columns and are generally easy for all the initialization methods. We solved 15 asymmetric instances in total. The average running times for the three initialization methods

Table 2. Normalized iterations and CPU time averages for symmetric TSPs

n	Bal		SS		AKP	
	niter	cpu	niter	cpu	niter	cpu
a280	3.497	3.499	1.684	1.811	1.00	1.00
ali535	1.902	1.968	1.575	1.721	1.00	1.00
att48	2.702	3.000	2.000	2.111	1.00	1.00
att532	3.761	3.709	1.923	2.029	1.00	1.00
berlin52	2.161	2.360	2.000	2.160	1.00	1.00
bier127	1.310	1.387	1.549	1.723	1.00	1.00
brazil58	2.484	2.786	2.048	2.286	1.00	1.00
burma14	1.938	3.994	1.813	3.495	1.00	1.00
ch130	4.267	3.878	2.467	2.528	1.00	1.00
ch150	3.673	3.502	1.993	2.037	1.00	1.00
d198	2.271	2.432	1.684	1.810	1.00	1.00
d493	2.796	2.826	1.615	1.744	1.00	1.00
d657	2.576	2.635	1.689	1.856	1.00	1.00
d1291	2.551	2.925	1.675	2.086	1.00	1.00
eil51	2.961	3.150	1.980	2.150	1.00	1.00
eil76	2.624	2.774	1.752	1.919	1.00	1.00
eil101	2.575	2.563	1.776	1.950	1.00	1.00
fl417	2.395	2.463	1.648	1.816	1.00	1.00
fl1400	1.757	1.778	1.183	1.237	1.00	1.00
gil262	2.858	2.901	1.884	1.996	1.00	1.00
gr96	2.874	3.020	1.773	1.910	1.00	1.00
gr137	3.129	3.281	1.781	1.949	1.00	1.00
gr202	2.122	2.203	1.766	1.927	1.00	1.00
gr229	3.944	3.707	2.022	2.119	1.00	1.00
gr431	3.225	3.330	1.723	1.857	1.00	1.00
gr666	2.867	2.863	1.594	1.722	1.00	1.00
kroA100	2.624	2.524	2.118	2.250	1.00	1.00
kroA150	3.027	2.760	2.000	2.041	1.00	1.00
kroA200	3.472	3.273	2.051	2.150	1.00	1.00
kroB100	3.161	3.049	2.218	2.244	1.00	1.00
kroB150	2.524	2.530	1.910	2.047	1.00	1.00
kroB200	3.961	3.636	2.056	2.122	1.00	1.00
kroC100	2.943	3.044	2.048	2.253	1.00	1.00
kroD100	2.491	2.526	1.981	2.105	1.00	1.00
kroE100	3.538	3.583	2.275	2.444	1.00	1.00
lin105	3.087	3.132	2.304	2.363	1.00	1.00
lin318	3.396	3.249	2.260	2.350	1.00	1.00
p654	3.263	3.551	1.910	2.284	1.00	1.00
pcb442	3.301	3.387	1.683	1.853	1.00	1.00
pr76	2.438	2.480	1.913	2.020	1.00	1.00
pr107	2.989	2.906	1.967	2.031	1.00	1.00
pr124	4.783	4.573	2.725	2.843	1.00	1.00
pr136	2.732	2.810	1.667	1.789	1.00	1.00
pr144	5.206	5.000	3.059	3.330	1.00	1.00
pr152	2.645	2.509	2.026	2.026	1.00	1.00
pr226	2.947	2.936	2.136	2.215	1.00	1.00
pr264	3.964	3.830	2.201	2.382	1.00	1.00
pr299	5.210	4.914	2.177	2.305	1.00	1.00
pr439	3.545	3.453	2.029	2.143	1.00	1.00
pr1002	3.766	3.655	1.840	1.955	1.00	1.00
rat99	3.444	3.528	1.855	1.934	1.00	1.00
rat195	3.746	3.786	1.979	2.063	1.00	1.00
rat575	4.525	4.298	1.860	1.919	1.00	1.00
rat783	5.024	4.639	2.036	2.190	1.00	1.00
rd100	2.629	2.762	1.819	1.933	1.00	1.00
rd400	4.003	3.609	2.028	2.099	1.00	1.00
rl1304	3.690	3.646	2.109	2.391	1.00	1.00
si175	2.047	2.140	1.618	1.762	1.00	1.00
si535	2.376	2.225	1.933	2.072	1.00	1.00
si1032	3.410	3.479	2.391	2.766	1.00	1.00
st70	2.667	2.531	1.987	1.918	1.00	1.00
swiss42	3.258	4.099	2.323	2.899	1.00	1.00
ts225	3.783	4.383	1.360	1.567	1.00	1.00
tsp225	4.032	3.947	1.968	2.035	1.00	1.00
u159	3.884	3.668	2.027	2.116	1.00	1.00
u574	4.516	4.283	1.962	2.052	1.00	1.00
u724	5.731	5.339	2.016	2.082	1.00	1.00
u1060	3.329	3.342	1.700	1.900	1.00	1.00
ulysses16	1.556	2.998	1.778	2.333	1.00	1.00
ulysses22	1.654	3.172	1.846	1.784	1.00	1.00
vm1084	3.957	3.802	1.978	2.175	1.00	1.00
Average	**3.176**	**3.238**	**1.940**	**2.105**		

(Bal, SS, AKP) are 0.186, 0.161 and 0.084 seconds respectively. From Table 3 we can observe that AKP performs better than Bal and SS on the asymmetric instances. In Table 4 we give the normalized ratios taken from Table 3.

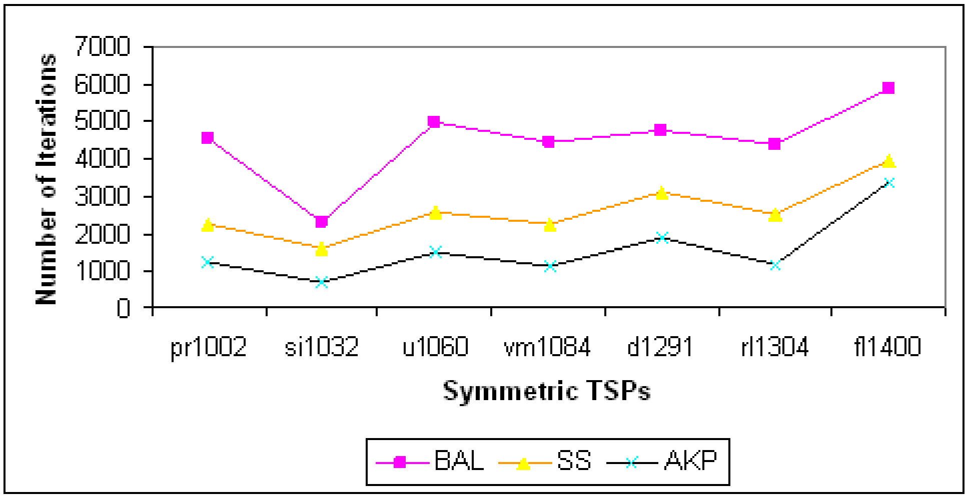

Figure 5. Number of iterations for the largest symmetric TSPs.

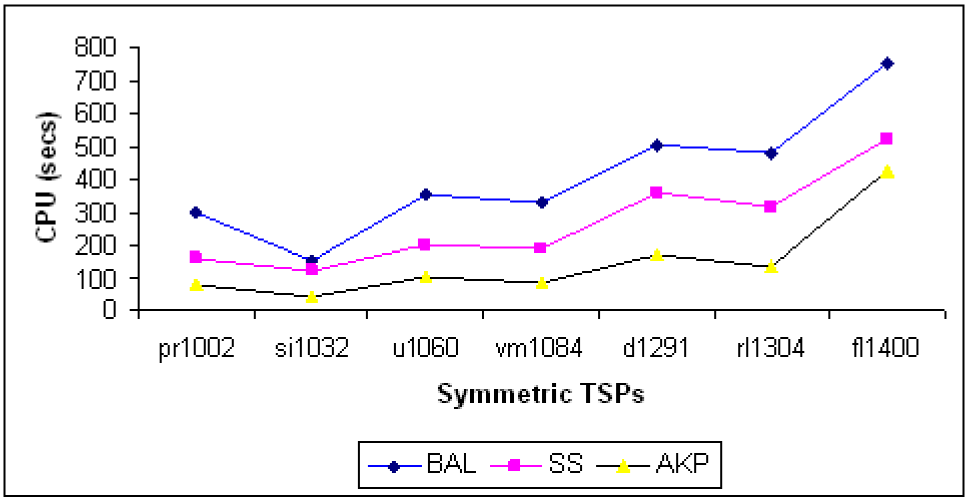

Figure 6. CPU time (in seconds) for the largest symmetric TSPs.

Table 3. TSPLib95: Asymmetric instances, (time in seconds)

name	n	zvalue	Bal		SS		AKP	
			niter	cpu	niter	cpu	niter	cpu
br17	17	0	23	0.009	23	0.011	10	0.006
ft53	53	5931	152	0.103	143	0.098	94	0.057
ft70	70	37,978	198	0.181	206	0.203	145	0.130
ftv33	34	1185	74	0.050	70	0.036	36	0.020
ftv35	36	1381	82	0.045	75	0.042	40	0.025
ftv38	39	1438	90	0.050	84	0.044	45	0.022
ftv44	45	1521	105	0.066	92	0.056	47	0.028
ftv47	48	1652	110	0.077	102	0.069	54	0.041
ftv55	56	1435	130	0.105	106	0.080	52	0.038
ftv64	65	1721	163	0.139	139	0.122	76	0.063
ftv70	71	1766	172	0.164	127	0.128	63	0.058
ftv170	171	2631	460	1.320	352	1.116	193	0.575
kro124p	100	33,978	285	0.392	193	0.298	99	0.144
p43	43	148	106	0.073	85	0.047	50	0.025
ry48p	48	12,517	146	0.010	83	0.059	36	0.028
Average			**153.067**	**0.186**	**125.333**	**0.161**	**69.333**	**0.084**

Table 4. Normalized iterations and CPU time averages for asymmetric TSPs

n	Bal		SS		AKP	
	niter	**cpu**	**niter**	**cpu**	**niter**	**cpu**
br17	2.300	1.440	2.300	1.744	1.00	1.00
ft53	1.617	1.803	1.521	1.721	1.00	1.00
ft70	1.366	1.398	1.421	1.566	1.00	1.00
ftv33	2.056	2.462	1.944	1.770	1.00	1.00
ftv35	2.050	1.812	1.875	1.688	1.00	1.00
ftv38	2.000	2.285	1.867	2.000	1.00	1.00
ftv44	2.234	2.333	1.957	2.000	1.00	1.00
ftv47	2.037	1.884	1.889	1.692	1.00	1.00
ftv55	2.500	2.792	2.038	2.125	1.00	1.00
ftv64	2.145	2.225	1.829	1.950	1.00	1.00
ftv70	2.730	2.838	2.016	2.216	1.00	1.00
ftv170	2.383	2.296	1.824	1.940	1.00	1.00
kro124p	2.879	2.728	1.949	2.076	1.00	1.00
p43	2.120	2.938	1.700	1.875	1.00	1.00
ry48p	4.056	0.355	2.306	2.111	1.00	1.00
Average	**2.298**	**2.106**	**1.896**	**1.898**		

4.2. *Benchmark APs (OR - Library)*

In this section we evaluate the performance of the three initialization methods in a collection of test data sets taken from OR - Library [21]. We choose eight (8) dense APs from this collection. These instances were proposed by Beasley [22]. The size of the instances range from 100 to 800 with step 100. Table 5 compares the number of iterations and the CPU time of the three initialization methods in the selected data set. The value of the optimal solution for each one of the instances is given in the third column of the Table 5. From the data in Table 5 we can clearly see that AKP is faster than the other two methods on all test instances. The average speedup of the AKP compared to Bal and SS is 2.821 and 1.338 times. Finally, in Table 6 we give the normalized ratios taken from Table 5.

Table 5. OR - Library instances, (time in seconds)

name	n	zvalue	Bal		SS		AKP	
			niter	**cpu**	**niter**	**cpu**	**niter**	**cpu**
assign100	100	305	383	0.053	263	0.040	179	0.025
assign200	200	475	1210	0.435	612	0.244	436	0.168
assign300	300	626	2162	1.534	1031	0.819	776	0.567
assign400	400	804	3302	3.869	1801	2.291	1355	1.645
assign500	500	991	5147	9.119	2309	4.291	1873	3.245
assign600	600	1176	7430	18.193	2738	7.198	2432	6.078
assign700	700	1362	9755	32.374	3198	11.321	2697	9.070
assign800	800	1552	12,980	54.684	3684	16.670	3692	15.871
Average			**5296.125**	**15.033**	**1954.500**	**5.359**	**1680.000**	**4.584**

Table 6. Normalized iterations and CPU time averages for OR - Library APs

n	Bal		SS		AKP	
	niter	**cpu**	**niter**	**cpu**	**niter**	**cpu**
assign100	2.140	2.106	1.469	1.608	1.00	1.00
assign200	2.775	2.587	1.404	1.450	1.00	1.00
assign300	2.786	2.705	1.329	1.444	1.00	1.00
assign400	2.437	2.352	1.329	1.393	1.00	1.00
assign500	2.748	2.810	1.233	1.322	1.00	1.00
assign600	3.055	2.993	1.126	1.184	1.00	1.00
assign700	3.617	3.569	1.186	1.248	1.00	1.00
assign800	3.516	3.445	0.998	1.050	1.00	1.00
Average	**2.884**	**2.821**	**1.259**	**1.338**		

5. Discussion on the Initialization Methods

From the computational results reported in section 4, we make the following observations: (1) The AKP initialization method is faster than the other two methods in terms of CPU time with a speedup varying between 1.3 and 3.4 and (2) The speed of AKP compared to the speed of the other two methods increases with instance sizes. But, which factors are responsible for the computational superiority of AKP initialization method? In this section, we give a theoretical explanation that reveals the superiority of AKP.

The most important factor that determines the computational efficiency of an algorithm for APs is the quality of the initial solution. In our case, EPSA uses as an initial solution the data structure of a tree T. All the consequent trees that are computed during EPSA's execution can be assigned a numerical value $\alpha(T)$. This value, called column level or stage of a solution T, denotes the "distance" of a solution T from the optimal solution. The iterations are grouped in stages. The last computed tree of the last stage is optimal. The optimal solution has always column level 0. Every iteration of the EPSA aims at reducing the stage number of the current solution. Let us now examine the column level of each one of the three initialization methods.

5.1. *Balinski tree*

To compute the column level of the Balinski tree we use the following procedure. For each column node j we define the "level degree" $\beta(j)$ which is computed as follows [23]:

$$\beta(j) = \begin{cases} d(j) - 2, if d(j) \geq 3 \\ 0, \ otherwise \end{cases} \tag{7}$$

The column level of the initial Balinski tree solution T_1 is then defined [23] as

$$\alpha(\mathrm{T}_1) = \sum_{j=1}^{n} \beta(j) \leq n - 1$$

Hence, EPSA using Balinski tree should pass through n levels in the worst case in order to reach the optimal solution.

5.2. *Simple start*

In [23] an algorithm for an mxn transportation problem that uses the same initial solution is described. In that paper, it is stated that the column level of the initial solution is $\sum_{i=1}^{n} b_i$, where b_i is the demand of column node i. Hence, in the AP we have that the column level of the simple start solution T_2 used by EPSA is always n, as the demand of all the column nodes in an assignment problem equals 1.

5.3. *AKP*

Finally, let us examine the column level of the AKP initial solution T_3. To compute the column level of the AKP tree we use the same procedure used for the Balinski tree. Again it is

$$\alpha(\mathrm{T}_3) = \sum_{j=1}^{n} \beta(j) \leq n - 1$$

where $\beta(j)$ is computed from Relation 7. It is easy to see that the upper bound $n-1$, in the previous relation, is only achieved when all the column nodes except for one have degree 1. This can be achieved only (1) if all the row nodes are connected with a single column node k and (2) if the input cost matrix C satisfies the relationship

$$c_{ik} = \min_{j=1,\ldots,n} \{c_{ij}\}$$

for all $i = 1, 2, \ldots, n$

Summarizing the above analysis we have that the column levels for the three initialization methods are

$$\alpha(T_1) = n \text{ and } \alpha(T_2) = \alpha(T_3) \leq n - 1$$

which shows that the AKP initialization method is more closer to the optimal solution.

Finally, given that the number of the stages of the second initialization method is always n, one would expect that EPSA using this method would take more iterations to terminate. This is something that does not hold in practice, as one can see from the experimental results in Section 4. This is due to the fact that the nature of the initial solution used by EPSA is not the same. The initialization methods SS and AKP use a forest, whereas Bal initialization method uses a tree. It is well known that a tree consists of a data structure difficult to handle. EPSA using Bal initialization method visits only strong trees which are obtained from strongly feasible trees by removing the feasibility requirement. In this case it is time expensive to determine the pair of leaving and entering edge involved in a single iteration. The other two initialization methods (SS and AKP) maintain and update a forest rather than a single tree. Hence, there are also other criteria to consider apart from the number of stages when analyzing the computational performance of the three initialization methods. In Table 7 we present the computed number of stages for the test data sets taken from OR - Library. The data in Table 7 shows that the third initialization method (AKP) has the smallest number of stages over all benchmark instances. One can observe from Table 7, that EPSA using SS initialization method always takes the maximum number of stages, which is equal to n. But, why SS initialization method always performs better than Bal initialization method? After carefully examination of the computational behavior of SS initialization method, we observed that SS initialization method always takes fewer iteration per stage than Bal method. Also, these iterations consist of the T^* tree which has few nodes. Specifically, T^* tree consists of one or two nodes. Also, the same result hold for all the instances used in our experimental study. Hence, by using the theory concerning the column level of the separate tree solutions, we can justify the computational efficiency of EPSA using the AKP initialization method.

Table 7. Number of stages for the OR - Library APs

name	**n**	**Bal**	**SS**	**AKP**
		stages	**stages**	**stages**
assign100	100	69	100	9
assign200	200	143	200	39
assign300	300	237	300	57
assign400	400	320	400	96
assign500	500	406	500	153
assign600	600	496	600	202
assign700	700	582	700	273
assign800	800	671	800	328

6. Conclusions

In this paper we presented a comparative computational study of three different initialization methods for the exterior point simplex algorithm. A crucial factor for the computational efficiency of algorithms for APs is the initialization method used. The computational efficiency of an initialization method depends on the distance between the initial solution and the optimal solution and on the structure of the initial solution.

From the experimental evaluation we obtain a precise ranking of the three initialization methods presented. The initialization method using Balinski's feasible tree (Bal) is the worst among the three compared methods on all test instances. On the other hand, we observe clearly the superiority of the initialization method AKP. In particular, AKP has the best performance on all benchmark assignment problems. The other initialization method (SS) is better than Bal but worst than AKP. On all instances it is the second best initialization method.

7. Acknowledgments

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] Kuhn, W.H., 1955, The Hungarian method for the assignment and transportation problems. *Naval Research Logistics Quarterly*, **2**, 83-97.

[2] Bertsekas, R.D., 1981, A new algorithm for the assignment problem. *Mathematical Programming* **21**, 152-171.

[3] Balinski, L.M., 1986, A competitive (dual) simplex method for the assignment problem. *Mathematical Programming*, **34**, 125-141.

[4] Goldfarb, D., 1985, Efficient dual simplex methods for the assignment problem. *Mathematical Programming*, **37**, 187-203.

[5] Akgül, M., 1988, A sequential dual simplex algorithm for the linear assignment problem. *Operations Research Letters*, **7**, 155-158.

[6] Paparrizos, K., 1991, A relaxation column signature method for assignment problems. *European Journal of Operational Research*, **50**, 211-219.

[7] Paparrizos, K., 1988, A non-dual signature method for the assignment problem and a generalization of the dual simplex method for the transportation problem. *RAIRO Operations Research*, **22**, 269-289.

[8] Akgül, M. and Ekin, O., 1991, A dual feasible forest algorithm for the assignment problem. *RAIRO Operations Research*, **25**, 403-411.

[9] Paparrizos, K., 1991, An infeasible (exterior point) simplex algorithm for assignment problems. *Mathematical Programming*, **51**, 45-54.

[10] Papamanthou, Ch., Paparrizos, K. and Samaras N., 2004, Computational experience with exterior point algorithms for the transportation problem. *Applied Mathematics and Computation*, **158**, pp. 459-475.

[11] Amico, D.M. and Toth, P., 2000, Algorithms and codes for dense assignment problems: the state of the art. *Discrete Applied Mathematics*, **100**, 17-48.

[12] Martello S. and Toth, P., 1987, Linear assignment problems, in Surveys in Combinatorial Optimization. In: S. Martello, G. Laporte, M. Minoux and C. Ribeiro, (Eds) (North-Holland, Amsterdam), *Annals of Discrete Mathematics* **31**, pp. 259-282.

[13] Derigs, U., 1985, The shortest augmenting path method for solving assignment problems - Motivation and computational experience, *Annals of Operations Research*, **4**, 57-102.

[14] Carpaneto G. and Toth, P., 1987, Primal-dual algorithms for the assignment problem. *Discrete Applied Mathematics*, **18**, 137-153.

[15] Kennington J. and Wang, Z., 1991, An Empirical Analysis of the Dense Assignment Problem: Sequential and Parallel Implementations. *ORSA Journal on Computing*, **3(4)**, 299-306.

[16] Volgenant, A., 1996, Linear and semi-assignment problems: A core oriented approach. *Computers and Operations Research*, **23**, 917-932.

[17] Achatz A., Paparrizos K., Samaras N. and K. Tsiplidis, 2002, A forest exterior point algorithm for assignment problems, in Combinatorial and Global Optimization, In: M.P. Pardalos, A. Midgalas and R. Buckard (Eds) (Word Scientific Publishing Co.) pp. 1-10.

[18] Achatz, H., Kleinschmidt P. and Paparrizos, K., 1991, A dual forest algorithm for the assignment problem. In: *DIMACS Series in Discrete Mathematics and Theoretical Computer Science (joint publication of the Association for Computing Machinery and the American Mathematical Society), Applied Geometry and Discrete Mathematics*, **4**, pp. 1-12.

[19] Papamanthou, Ch., Paparrizos, K. and Samaras N., 2005, A Parametric Visualization Software for the Assignment Problem. *Yugoslav Journal of Operations Research*, **15(1)**, pp. 1-12.

[20] Reinelt, G., 1991, TSPLib - A Traveling Salesman Problem Library. *ORSA Journal on Computing*, **3**, 376-384.

[21] Beasley, J.E., 1990, OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, **41**, 1069-1072.

[22] Beasley, J.E., 1990, Linear programming on cray supercomputers. *Journal of Operational Research Society*, **41**, 133-139.

[23] Paparrizos, K., 1996, A non improving simplex algorithm for transportation problems. *RAIRO Operations Research*, **30**, 1-15.