Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER Applied Mathematics and Computation 158 (2004) 459–475

APPLIED MATHEMATICS AND COMPUTATION

www.elsevier.com/locate/amc

# Computational experience with exterior point algorithms for the transportation problem

Charalampos Papamanthou, Konstantinos Paparrizos, Nikolaos Samaras [*]

*Department of Applied Informatics, University of Macedonia, 156 Egnatia Str., Thessaloniki 54006, Greece*

**Abstract**

An experimental computational study to compare the classical primal simplex algorithm and the exterior point algorithms for the transportation problem (TP) is presented. Totally, four algorithms are compared on uniformly randomly generated test problems. The results are very encouraging for one of the competitive algorithms. In particular, a dual forest exterior point algorithm is on average up to 4.5 times faster than network simplex algorithm on TPs of size $300 \times 300$ and for all classes. This result leads into corresponding savings in computational time. From the computational performance we conclude that as the problem size increases, exterior point algorithm get relatively faster.

*Keywords:* Transportation problem; Network simplex algorithm; Exterior point simplex algorithms; Algorithm evaluation; Experimental computational study

## 1. Introduction

The transportation problem (TP) is one of the fundamental problems of network flow optimization. A surprisingly enough large number of real-life

[*] Corresponding author.
*E-mail addresses:* it0067@uom.gr (C. Papamanthou), paparriz@uom.gr (K. Paparrizos), samaras@uom.gr (N. Samaras).

0096-3003/$ - see front matter © 2003 Elsevier Inc. All rights reserved.
doi:10.1016/j.amc.2003.09.028

applications can be formulated as a TP [3,7,19–21]. The TP is a generalization of the assignment problem, where the right-hand side of the constraint may take positive integer values and the decision variables any non-negative integer. An extension of the TP is the transshipment problem whereby the network is not bipartite.

Transportation models have become quite popular during the past years. TP is a network structured linear programming problem that has deservedly received an important attention in the literature. Dantzig [9] adapted the primal simplex algorithm (PSA) to solve the TP. The relationship between basic solutions in the TP and the tree structure of a graph presented by Koopmans [10]. An alternate method to stepping-stone [6] for solving TPs proposed by Arshom and Khan [4]. Orlin et al. [12] developed a polynomial dual network simplex pivot rule modifying the strongly polynomial capacity scaling algorithms for the transshipment problem. Their best pivoting strategy leads to an $O(m^3 \log n)$ bound on the number of pivots. Portugal et al. [18] developed a preconditioner based on an incomplete QR decomposition for use in interior point methods to solve TPs. A method that can compute the total cost bounds of the TP, where at least one of the supply or demand is varying, proposed by Liu [11].

Recently, a new category of algorithms for network flow problems [2,14,17] and the general linear programming problem [16] have been developed. This category is called exterior point simplex algorithm (EPSA). The dual in nature EPSAs are initialized with a dual feasible basic solution. Contrary to the dual simplex algorithm, EPSAs destroy dual feasibility in intermediate iterations. Dual feasibility is restored again at optimal solution. A non-improving simplex type algorithm for the TP has been developed by Paparrizos [15]. Preliminary computational results on assignment problems demonstrate the superiority of EPSA over the classical PSA [1,13]. Particularly, a dual forest exterior point algorithm is about 3 times faster than simplex in terms of CPU time and up to 6 times faster in terms of number of iterations on randomly generated assignment problems of size $300 \times 300$.

In this paper an experimental computational study on randomly generated full dense TPs is presented to establish the practical value of the EPSAs. For the first time in the literature the EPSA is tested on TPs, using three different initialization techniques. The computational efficiency of algorithms for TPs is highly dependant on the initial solution. Four algorithms take part in our experimental investigation. These are: (i) The PSA [5,8], (ii) The EPSA starting with Balinski's feasible tree [17] (BFT), (iii) The EPSA starting with a simple forest [15] (SSF) that is neither primal nor dual feasible and (iv) The EPSA starting with a feasible forest [1] (AKP). The worst-case time complexity of the AKP algorithm is $O(\mu(m+n)D)$ where $\mu = \min\{m, n\}$ and $D$ is the total demand. Also, AKP solves a TP in at most $\mu D - \mu(\mu - 1)/2$ iterations. The experimental results indicate that the AKP is the faster algorithm.

The plan of the paper is as follows. A short outline of EPSA and the different initialization techniques is given in Section 2. An illustrative example presented in Section 3. In Section 4 we report about our experimental investigation on randomly generated TPs. Conclusions are presented and future research is outlined in Section 5.

## 2. Exterior point algorithms

A TP can be represented by bipartite graph $G(S,D,E) = G(N,E)$, where $S$, $D$ are two discrete sets of nodes such $|S| = m$, $|D| = n$ and $N = S \cup D$. Notations $|S|$ and $|D|$ stands for the cardinality number of the sets $S$ and $D$ respectively. Here, $E$ is the arc set. An arc $(i,j) \in E$ is directed from nodes of $S$ to nodes of $D$. Supply nodes denoted by $i \in S$, whereas demand nodes denoted by $j \in D$.

The mathematical formulation of the TP with an $m \times n$ cost matrix $c$, an $m \times 1$ supply vector $a$ and an $n \times 1$ demand vector $b$ such as $\sum_{i \in S} a(i) = \sum_{j \in D} b(j)$ is the following:

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

$$\text{s.t.} \quad \sum_{j=1}^{n} x_{ij} = a(i), \quad i \in S \tag{2}$$

$$\sum_{i=1}^{m} x_{ij} = b(j), \quad j \in D \tag{3}$$

$$x_{ij} \geqslant 0, \quad i \in S, \ j \in D. \tag{4}$$

In the relations (1)–(4), $x_{ij}$ is the number of the units that are transported from the supply node $i$ to the demand node $j$. The associated dual problem to (PTP) is the following

$$\text{(DTP)}\ \max \quad \sum_{i=1}^{m} u_i + \sum_{j=1}^{n} v_j$$

$$\text{s.t.} \quad u_i + v_j \leqslant c_{ij} \quad 1 \leqslant i \leqslant m, \ 1 \leqslant j \leqslant n.$$

Given a pair of feasible solutions $x$ and $(u,v)$ for the problems (PTP) and (DTP) respectively, the complementary slackness condition is stated as $x_{ij}s_{ij} = 0$, $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant n$ where

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

is the reduced cost corresponding to the primal variable $x_{ij}$.

The PSA for TPs is well known. The main procedure of EPSA is as follows. At each iteration a special solution $T$ is computed. Structure $T$ is depicted as a rooted tree of the bipartite graph. The root can be either a row or a column node. Using the current solution $T$, the set of nodes are partitioned into two subsets $F$ and $T \sim F$. The EPSA stops when $F$ becomes void. As EPSAs are of dual simplex type, 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 $s_{gh} = \min\{s_{ij} : i \in F, j \in T \sim F\}$. The entrance of the arc $(g, h)$ always forms a circle $C$ which is always partitioned into two main subsets, $C^+$ and $C^-$. Subset $C^+$ contains the arcs having the same direction with the entering arc $(g, h)$, while for the subset $C^-$ always is $C^- = C \sim C^+$.

Then the leaving arc $(k, l)$ is chosen. The three EPSAs considered in this paper differ among each other in the way they are initialized. As a consequence, the leaving arc is chosen in different ways. One special data structure that the algorithms use 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.

A brief description of the main procedure of EPSA is presented in the following pseudocode. We use the notation $\langle \cdots \rangle$ to embrace explanation comments.

*Exterior point algorithm*

```
Step 0. (Initialization). Start with a special solution $T$. Determine the subsets
$F$, $T \sim F$ and $s_{ij}$ according to the initialization method.
Step 1. (General step).
   while ($F \neq \emptyset$) $\langle$if $F = \emptyset$, STOP, tree $T$ is optimal$\rangle$
     $\delta = s_{gh} = \min\{s_{ij} : i \in F \wedge j \in T \sim F\}$ $\langle$arc $(g, h)$ is the entering arc$\rangle$
     Choose the leaving arc $(k, l)$ $\langle$using the special rules of each algorithm$\rangle$
     $T' = T \cup (g, h) \sim (k, l)$ $\langle$update tree $T\rangle$
     if $h \in T^*$ $\langle$update $s_{ij}\rangle$
        for each row node $i \in T^*$ set $s_{i.}(T') = s_{i.}(T) + \delta e^T$
        for each column node $j \in T^*$ set $s_{.j}(T') = s_{.j}(T) + \delta e^T$
     $\langle$update $F$ and $T \sim F\rangle$
     set $T = T'$
end
```

Following this pseudocode we present three different initialization methods for the EPSAs. These methods are (i) the Balinski tree, (ii) the simple start forest and (iii) the AKP forest. In the following figures we draw supply nodes as circles and demand nodes as squares.

### 2.1. The Balinski tree initialization method

BFT uses the Balinski tree as a starting solution. The Balinski tree is of dual nature. It is not primal feasible. It is always rooted on row node 1 and all the other column nodes lay one level beneath. The remaining row nodes always lie at depth 2 (regarding the depth of the root of the tree equal to 0). The Balinski tree has always $n$ fixed arcs of type $(1,j)$, where $j$ is a column node. If $T$ is the Balinski tree, we can easily compute the main variables (i.e. dual variables and reduced costs) of $T$. Initially we set $u_1(T) = 0$ and $v_j(T) = c_{1j}$, $j = 1, 2, \dots, n$. Thus, for each arc $(1,j)$ we have $s_{1j}(T) = c_{1j} - u_1(T) - v_j(T) = 0$. If we know a row index $i$, the respective column index associated with the minimum difference $c_{ij} - v_j$ of row $i$ can easily be computed by setting

$$j(i) = \operatorname{argmin}\{c_{ij} - v_j : j = 1, 2, \dots, n\}.$$

If now we set

$$u_i(T) = c_{ij(i)} - v_{j(i)}, \quad i = 2, 3, \dots, m$$

then we can add a single arc to the tree and thus $(i, j(i)) \in T$. Having computed all the basic arcs of the Balinski tree, it is now easy to compute the reduced costs of the non-basic arcs by relation (5).

The decision variables $x_{ij}$ corresponding to basic arcs $(i,j)$ of the tree are computed as follows. As the Balinski tree always consists of three levels, 0, 1 and 2 and all the row nodes except for the root are leaves, we set $x_{ij}(T) = a(i)$, $i \neq 1$, $(i,j) \in T$. For the remaining basic arcs of the tree which are of type $(1,j)$, we set $x_{1j}(T) = b(j)$, if $j$ is a leaf of the tree, while when $j$ is not a leaf of the tree we set

$$x_{1j}(T) = b(j) - \sum_{(i,j) \in T,\, i \neq 1} x_{ij}.$$

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, $T \sim F$ is a subtree of $T$. A general Balinski tree for a TP of size $m \times n$ can be seen at Fig. 1.

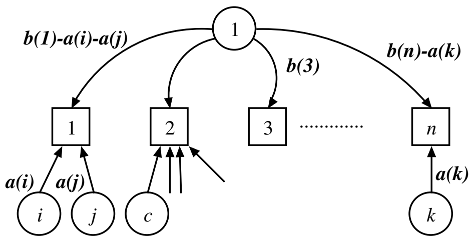

Fig. 1. The general Balinski tree for an $(m \times n)$ TP.

## *2.2. The simple start forest initialization method*

SSF starts its execution with a very simply constructed forest, the simple start forest. Let $Q$ be the simple start forest. Forest $Q$ consists of the $m+n$ isolated nodes. To determine the sets $F$, $T \sim F$, it is set $F = \{i : i \in S\}$ and $T \sim F = \{j : j \in D\}$. The dual variables of both the row ($u_i$, $i = 1, \ldots, m$) and column ($v_j$, $j = 1, \ldots, m$) nodes are set to zero. Thus, by relation (5), we have $s_{ij}(Q) = c_{ij}$. A general simple start forest for a TP of dimensions $m \times n$ can be seen in Fig. 2.

As the EPSAs work with rooted trees, we have to provide a root to the simple start forest and thus convert the forest $Q$ to a tree $T$. This can easily be done by inserting an artificial node 0 (which is the root of tree $T$) and adding $m+n$ artificial arcs. Therefore, we add $m$ artificial arcs $(i, 0)$ for each row node $i$ and $n$ artificial arcs $(0, j)$ for each row node $j$. The unit costs $c_{ij}$ of the artificial arcs are set equal to zero.

All the basic decision variables of type $x_{i0}$, $i = 1, 2, \ldots, m$ are initially set equal to $a(i)$, while all the decision variables of type $x_{0j}$, $j = 1, 2, \ldots, n$ are initially set equal to $b(j)$. SSF updates the sets $F$, $T \sim F$ in way that the following relations are satisfied:

$$F = \{T_i : i \in S\}, \quad T \sim F = \{T_j : j \in D\}.$$

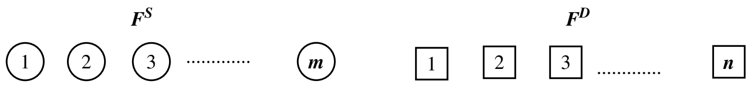

Fig. 2. The general simple start forest for an $(m \times n)$ TP.

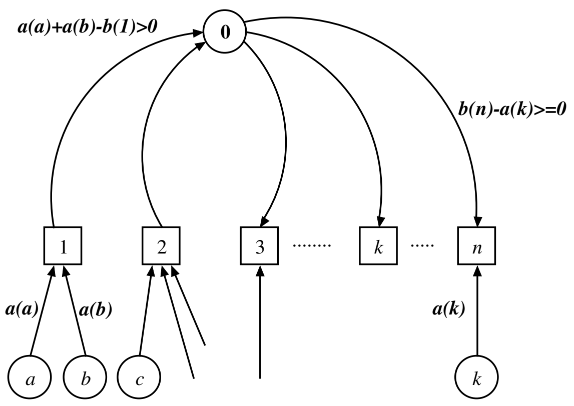

Fig. 3. The general AKP forest for an $(m \times n)$ TP.

### 2.3. The AKP forest initialization method

The forest AKP is the starting solution for AKP algorithm. It is the main reason for the computational efficiency of this algorithm. In order to compute the vectors $u$, $v$ of the dual variables of the row and the column nodes respectively, we set

$$v_j(Q) = 0, \quad \forall j \in D \tag{6}$$

and

$$u_i(Q) = \min\{c_{ij} : j = 1, \ldots, n\}, \quad \forall i \in S. \tag{7}$$

The arc $(i, t)$ will be a basic arc of the forest, where $t$ is a column node derived from the relation $u_i(Q) = c_{it}$. For each arc $(i, t)$ we set $x_{it} = a(i)$. The reduced costs $s_{ij}$ can now easily be computed, using the relation (5).

Note that $s_{ij}(Q) \geqslant 0$, which means that forest $Q$ is dual feasible. The initial set $F$, is

$$F = \left\{ T_j : b_j < \sum_{i \in T_j} a_i \right\}. \tag{8}$$

Accordingly,

$$T \sim F = \left\{ T_j : \sum_{i \in T_j} a_i \leqslant b_j \right\}. \tag{9}$$

For the same reason referred at the simple start method, we need to transform the forest $Q$ to a tree $T$. This can easily 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

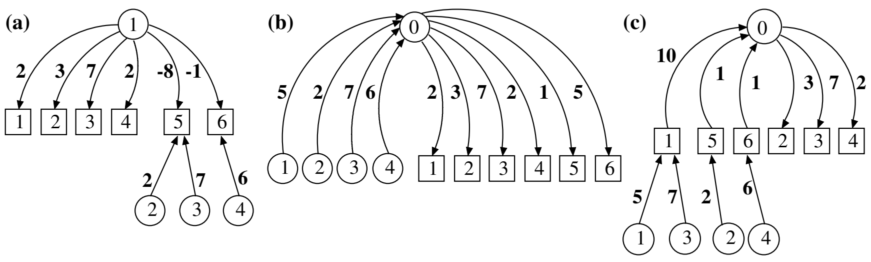

Fig. 4. Starting solutions: (a) Balinski, (b) simple start, (c) AKP.

$$x_{j0} = \sum_{i \in T_j} a_i - b_j > 0$$

and

$$x_{0j} = b_j - \sum_{i \in T_j} a_i \geqslant 0.$$

A general AKP tree for a TP of dimensions $m \times n$ is illustrated in Fig. 3.

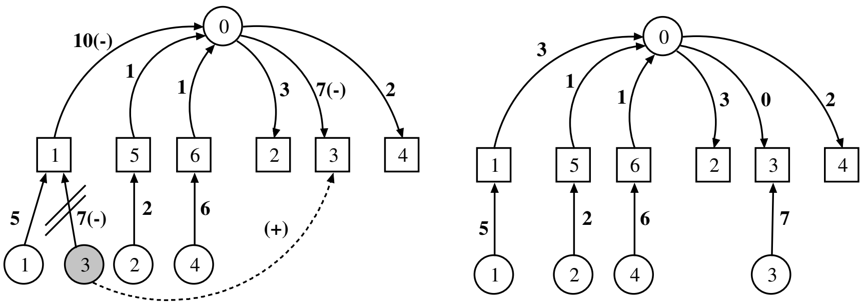

Fig. 5. The first iteration of the SSF algorithm.

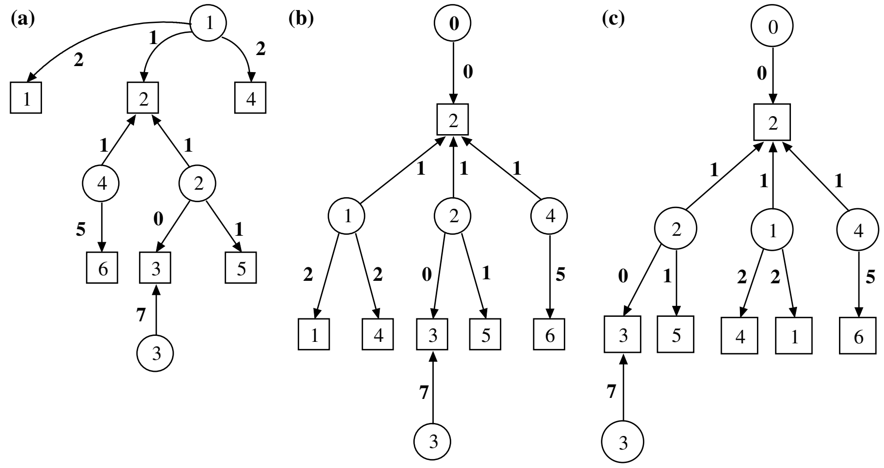

Fig. 6. Optimal solutions: (a) BFT, (b) SSF and (c) AKP.

## 3. An illustrative example

If we apply the three initialization methods for the $4\times 6$ cost matrix

$$C=\begin{bmatrix} -5 & 49 & 28 & 20 & 47 & 43 \\ 19 & 62 & 32 & 61 & 0 & 13 \\ -2 & 63 & 14 & 46 & 28 & 53 \\ 1 & 50 & 41 & 26 & 59 & 0 \end{bmatrix}$$

the supply vector $a=[5 \quad 2 \quad 7 \quad 6]$ and the demand vector $b=[2 \quad 3 \quad 7 \quad 2 \quad 1 \quad 5]$, the reader can verify that the starting feasible solutions will be those of Fig. 4.

Following, we will present a complete iteration of the SSF algorithm. From relations (6) and (7) we can compute the vectors $u$, $v$ that belong to the dual

Table 1
Computational results for class 1 TPs

| Size | PSA | | BFT | | SSF | | AKP | |
|---|---|---|---|---|---|---|---|---|
| | niter | cpu | niter | cpu | niter | cpu | niter | cpu |
| $n\times n$ | | | | | | | | |
| 50 | 273,3 | 7,04 | 152,1 | 6,28 | 137,0 | 4,97 | 89,3 | 3,15 |
| 75 | 504,5 | 26,77 | 285,0 | 23,56 | 226,9 | 15,30 | 151,9 | 9,41 |
| 100 | 752,5 | 55,32 | 410,4 | 54,75 | 319,4 | 34,33 | 223,2 | 20,08 |
| 125 | 1.035,0 | 117,71 | 588,0 | 114,83 | 392,5 | 61,03 | 262,7 | 36,32 |
| 150 | 1.354,3 | 204,76 | 765,4 | 208,26 | 483,7 | 103,84 | 331,0 | 62,77 |
| 175 | 1.694,1 | 338,58 | 895,3 | 315,85 | 558,3 | 154,18 | 378,3 | 90,02 |
| 200 | 2.007,4 | 513,05 | 1.135,5 | 511,57 | 656,7 | 230,98 | 457,2 | 131,53 |
| 225 | 2.372,3 | 907,01 | 1.427,8 | 821,73 | 726,6 | 321,48 | 497,4 | 184,97 |
| 250 | 2.795,0 | 1.267,27 | 1.566,6 | 1.069,54 | 842,4 | 440,51 | 591,6 | 257,50 |
| 275 | 3.226,5 | 1.499,80 | 1.775,5 | 1.448,67 | 930,1 | 586,71 | 640,7 | 336,20 |
| 300 | 3.575,0 | 1.957,07 | 2.050,1 | 1.994,09 | 1.038,7 | 771,86 | 724,5 | 441,18 |
| $n\times 2n$ | | | | | | | | |
| 50 | 469,2 | 23,14 | 256,3 | 19,75 | 235,8 | 15,85 | 132,6 | 7,88 |
| 75 | 861,2 | 85,59 | 481,8 | 75,21 | 358,8 | 47,01 | 212,8 | 24,02 |
| 100 | 1.284,5 | 215,66 | 727,6 | 188,77 | 504,6 | 106,90 | 293,2 | 52,51 |
| 125 | 1.742,7 | 436,04 | 1.003,6 | 390,68 | 637,5 | 197,06 | 381,3 | 97,23 |
| 150 | 2.303,8 | 809,43 | 1.325,8 | 710,91 | 759,1 | 334,63 | 450,1 | 162,74 |
| 175 | 2.875,0 | 1.349,90 | 1.618,3 | 1.167,29 | 951,5 | 548,82 | 581,1 | 268,61 |
| 200 | 3.407,8 | 2.032,94 | 2.039,3 | 1.891,30 | 1.097,0 | 801,55 | 669,1 | 392,03 |
| $2n\times n$ | | | | | | | | |
| 50 | 476,4 | 23,08 | 264,0 | 20,80 | 221,8 | 14,39 | 171,4 | 9,62 |
| 75 | 833,3 | 71,80 | 466,4 | 73,73 | 357,8 | 43,71 | 288,0 | 31,18 |
| 100 | 1.257,6 | 178,01 | 700,9 | 184,83 | 480,8 | 94,01 | 383,5 | 68,20 |
| 125 | 1.749,9 | 371,35 | 989,9 | 388,82 | 644,6 | 184,40 | 505,8 | 130,46 |
| 150 | 2.307,1 | 686,71 | 1.208,2 | 665,98 | 743,4 | 299,14 | 597,4 | 212,60 |
| 175 | 2.832,9 | 1.326,01 | 1.219,2 | 725,30 | 906,6 | 479,68 | 725,3 | 334,15 |
| 200 | 3.443,1 | 2.071,15 | 1.982,2 | 1.888,26 | 1.079,7 | 703,67 | 884,3 | 512,54 |

Table 2
Computational results for class 2 TPs

| Size | PSA | | BFT | | SSF | | AKP | |
|---|---|---|---|---|---|---|---|---|
| | niter | cpu | niter | cpu | niter | cpu | niter | cpu |
| $n \times n$ | | | | | | | | |
| 50 | 261,0 | 6,79 | 155,9 | 6,20 | 140,1 | 4,97 | 89,2 | 3,04 |
| 75 | 506,7 | 23,61 | 273,0 | 21,56 | 224,6 | 15,16 | 151,7 | 9,39 |
| 100 | 757,6 | 55,25 | 426,3 | 55,04 | 312,1 | 32,78 | 214,8 | 20,42 |
| 125 | 1.034,2 | 114,99 | 559,9 | 106,29 | 377,1 | 58,72 | 252,1 | 33,71 |
| 150 | 1.340,2 | 204,06 | 717,6 | 188,16 | 474,6 | 99,89 | 325,5 | 61,27 |
| 175 | 1.660,0 | 332,72 | 921,8 | 330,19 | 540,5 | 151,47 | 364,5 | 87,41 |
| 200 | 2.059,8 | 628,44 | 1.083,4 | 481,85 | 655,3 | 230,00 | 449,6 | 135,21 |
| 225 | 2.451,7 | 764,47 | 1.359,4 | 757,38 | 772,2 | 333,37 | 541,4 | 200,91 |
| 250 | 2.811,0 | 1.294,36 | 1.516,2 | 1.035,18 | 848,3 | 450,92 | 572,8 | 253,61 |
| 275 | 3.152,3 | 1.731,11 | 1.821,7 | 1.495,49 | 893,2 | 562,84 | 625,9 | 329,85 |
| 300 | 3.554,2 | 2.270,05 | 1.969,5 | 1.897,86 | 1.031,2 | 763,10 | 722,6 | 448,69 |
| $n \times 2n$ | | | | | | | | |
| 50 | 451,3 | 22,29 | 261,5 | 20,02 | 232,4 | 15,31 | 135,4 | 7,82 |
| 75 | 823,9 | 86,96 | 490,8 | 77,78 | 362,4 | 47,14 | 222,1 | 24,01 |
| 100 | 1.279,4 | 224,19 | 702,0 | 184,78 | 509,5 | 110,23 | 294,8 | 50,83 |
| 125 | 1.750,8 | 467,18 | 989,1 | 395,21 | 617,8 | 199,27 | 363,8 | 90,34 |
| 150 | 2.278,1 | 804,31 | 1.329,2 | 730,33 | 770,0 | 335,47 | 459,2 | 163,15 |
| 175 | 2.857,1 | 1.300,25 | 1.573,4 | 1.151,28 | 942,6 | 538,71 | 570,3 | 258,46 |
| 200 | 3.486,7 | 1.783,20 | 1.958,1 | 1.831,08 | 1.095,8 | 818,45 | 673,4 | 395,62 |
| $2n \times n$ | | | | | | | | |
| 50 | 469,2 | 24,63 | 263,2 | 20,40 | 225,5 | 13,97 | 172,5 | 9,85 |
| 75 | 845,4 | 84,51 | 450,2 | 71,32 | 353,5 | 42,48 | 285,6 | 30,73 |
| 100 | 1.296,9 | 234,03 | 694,5 | 184,52 | 484,0 | 96,19 | 378,1 | 65,61 |
| 125 | 1.769,0 | 445,37 | 962,7 | 385,41 | 603,1 | 175,56 | 480,1 | 121,65 |
| 150 | 2.325,6 | 712,05 | 1.210,9 | 671,84 | 733,2 | 311,18 | 589,7 | 224,49 |
| 175 | 2.890,8 | 1.412,21 | 1.647,3 | 1.208,47 | 941,0 | 541,70 | 595,2 | 280,12 |
| 200 | 3.441,2 | 1.737,13 | 2.050,8 | 1.940,25 | 1.085,3 | 793,20 | 673,8 | 401,21 |

variables of the supply and the demand respectively. Thus, $u = [-5 \quad 0 \quad -2 \quad 0]$ and $v = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$. Additionally, by relation (5) we get

$$s = \begin{bmatrix} 0 & 54 & 33 & 25 & 52 & 48 \\ 19 & 62 & 32 & 61 & 0 & 13 \\ 0 & 65 & 16 & 48 & 30 & 55 \\ 1 & 50 & 41 & 26 & 59 & 0 \end{bmatrix}.$$

Finally, by relations (8) and (9) we compute the components $F$, $T \sim F$. Thus $F = \{T_1, T_5, T_6\} = \{1, 2, 3, 4\}$ and $T \sim F = \{T_2, T_3, T_4\} = \{2, 3, 4\}$. Note that $F$ contains only row nodes, while $T \sim F$ contains only column nodes. Now we have all the necessary data for the algorithm to work.

We determine the entering arc by setting $\delta = s_{33} = \min\{s_{ij} : i \in \{1, 2, 3, 4\} \wedge j \in \{2, 3, 4\}\}$. As we can see in Fig. 5, it is $C^- = \{(3, 1), (1, 0), (0, 3)\}$. The

Table 3
Computational results for class 3 TPs

| Size | PSA | | BFT | | SSF | | AKP | |
|---|---|---|---|---|---|---|---|---|
| | niter | cpu | niter | cpu | niter | cpu | niter | cpu |
| $n \times n$ | | | | | | | | |
| 50 | 269,5 | 6,95 | 151,8 | 6,08 | 136,6 | 4,82 | 89,0 | 3,04 |
| 75 | 508,1 | 26,66 | 283,9 | 22,72 | 226,2 | 14,98 | 151,4 | 9,26 |
| 100 | 759,5 | 55,17 | 410,0 | 53,20 | 320,5 | 33,93 | 223,8 | 21,51 |
| 125 | 1.032,7 | 113,70 | 587,1 | 111,02 | 391,0 | 60,22 | 262,2 | 35,77 |
| 150 | 1.359,5 | 241,72 | 764,8 | 208,35 | 483,1 | 102,79 | 329,8 | 60,68 |
| 175 | 1.712,1 | 408,23 | 896,7 | 320,65 | 558,5 | 155,79 | 377,2 | 88,20 |
| 200 | 2.000,2 | 605,07 | 1.135,7 | 520,46 | 655,8 | 231,44 | 458,1 | 137,50 |
| 225 | 2.325,9 | 871,59 | 1.318,4 | 764,67 | 749,1 | 324,08 | 519,7 | 184,25 |
| 250 | 2.789,2 | 1.274,09 | 1.563,7 | 1.068,19 | 842,3 | 439,00 | 591,1 | 256,80 |
| 275 | 3.165,5 | 1.753,92 | 1.616,0 | 1.338,75 | 921,0 | 585,70 | 621,5 | 327,69 |
| 300 | 3.594,2 | 2.231,47 | 1.938,4 | 1.867,91 | 1.036,7 | 778,83 | 718,8 | 440,35 |
| $n \times 2n$ | | | | | | | | |
| 50 | 256,1 | 10,38 | 361,2 | 26,39 | 267,2 | 13,30 | 218,9 | 10,11 |
| 75 | 514,5 | 37,43 | 676,9 | 101,73 | 412,7 | 38,02 | 339,4 | 30,46 |
| 100 | 818,7 | 96,49 | 999,9 | 255,03 | 572,3 | 83,51 | 475,0 | 67,82 |
| 125 | 1.090,4 | 196,85 | 1.442,5 | 556,03 | 735,5 | 154,34 | 613,3 | 125,54 |
| 150 | 1.542,5 | 387,42 | 1.857,5 | 1.016,40 | 877,4 | 253,09 | 732,0 | 204,27 |
| 175 | 1.856,3 | 821,34 | 2.518,9 | 1.759,50 | 1.072,5 | 394,49 | 857,6 | 314,91 |
| 200 | 2.309,6 | 1.209,96 | 3.028,5 | 2.561,97 | 1.233,0 | 584,77 | 1.037,0 | 467,33 |
| $2n \times n$ | | | | | | | | |
| 50 | 250,2 | 9,94 | 243,3 | 16,27 | 268,8 | 14,52 | 182,4 | 8,33 |
| 75 | 497,8 | 31,48 | 448,2 | 62,03 | 428,4 | 43,85 | 297,9 | 25,65 |
| 100 | 796,9 | 106,19 | 622,6 | 145,41 | 579,3 | 95,49 | 403,7 | 54,21 |
| 125 | 1.138,1 | 230,14 | 840,5 | 295,94 | 741,2 | 178,18 | 527,8 | 102,81 |
| 150 | 1.561,6 | 443,12 | 1.020,8 | 510,30 | 901,1 | 301,06 | 643,2 | 171,62 |
| 175 | 2.045,2 | 698,63 | 1.218,3 | 781,29 | 1.061,4 | 459,66 | 761,8 | 260,31 |
| 200 | 2.701,4 | 995,08 | 1.478,2 | 1.115,48 | 1.323,1 | 655,04 | 877,3 | 367,70 |

algorithm chooses the leaving arc $(k,l)$ by setting $\varepsilon = x_{k\ell} = \min\{x_{ij}(T) : (i,j) \in C^-\} \Rightarrow (k,\ell) \in \{(3,1),(0,3)\}$.

The algorithm uses the well known Cunningham rule [8] and finally sets $(k,l) = (3,1)$. The update of the values $x_{ij}$ is achieved by adding $\varepsilon$ units to the arcs belonging to $C^+$ and by subtracting $\varepsilon$ units to the remaining arcs. This can be seen in Fig. 5. After the update of the matrix $s$ according to the algorithm pseudocode, we get

$$s = \begin{bmatrix} 0 & 54 & 33 & 25 & 52 & 48 \\ 19 & 62 & 32 & 61 & 0 & 13 \\ -16 & 49 & 0 & 32 & 14 & 39 \\ 1 & 50 & 41 & 26 & 59 & 0 \end{bmatrix}.$$

Table 4
Normalized iterations and CPU time averages for TPs of class 1

| Size | PSA | | BFT | | SSF | | AKP | |
|---|---|---|---|---|---|---|---|---|
| | niter | cpu | niter | cpu | niter | cpu | niter | cpu |
| $n \times n$ | | | | | | | | |
| 50 | 3,06 | 2,23 | 1,70 | 1,99 | 1,53 | 1,58 | 1,00 | 1,00 |
| 75 | 3,32 | 2,84 | 1,88 | 2,50 | 1,49 | 1,63 | 1,00 | 1,00 |
| 100 | 3,37 | 2,75 | 1,84 | 2,73 | 1,43 | 1,71 | 1,00 | 1,00 |
| 125 | 3,94 | 3,24 | 2,24 | 3,16 | 1,49 | 1,68 | 1,00 | 1,00 |
| 150 | 4,09 | 3,26 | 2,31 | 3,32 | 1,46 | 1,65 | 1,00 | 1,00 |
| 175 | 4,48 | 3,76 | 2,37 | 3,51 | 1,48 | 1,71 | 1,00 | 1,00 |
| 200 | 4,39 | 3,90 | 2,48 | 3,89 | 1,44 | 1,76 | 1,00 | 1,00 |
| 225 | 4,77 | 4,90 | 2,87 | 4,44 | 1,46 | 1,74 | 1,00 | 1,00 |
| 250 | 4,72 | 4,92 | 2,65 | 4,15 | 1,42 | 1,71 | 1,00 | 1,00 |
| 275 | 5,04 | 4,46 | 2,77 | 4,31 | 1,45 | 1,75 | 1,00 | 1,00 |
| 300 | 4,93 | 4,44 | 2,83 | 4,52 | 1,43 | 1,75 | 1,00 | 1,00 |
| $n \times 2n$ | | | | | | | | |
| 50 | 3,54 | 2,94 | 1,93 | 2,51 | 1,78 | 2,01 | 1,00 | 1,00 |
| 75 | 4,05 | 3,56 | 2,26 | 3,13 | 1,69 | 1,96 | 1,00 | 1,00 |
| 100 | 4,38 | 4,11 | 2,48 | 3,59 | 1,72 | 2,04 | 1,00 | 1,00 |
| 125 | 4,57 | 4,48 | 2,63 | 4,02 | 1,67 | 2,03 | 1,00 | 1,00 |
| 150 | 5,12 | 4,97 | 2,95 | 4,37 | 1,69 | 2,06 | 1,00 | 1,00 |
| 175 | 4,95 | 5,03 | 2,78 | 4,35 | 1,64 | 2,04 | 1,00 | 1,00 |
| 200 | 5,09 | 5,19 | 3,05 | 4,82 | 1,64 | 2,04 | 1,00 | 1,00 |
| $2n \times n$ | | | | | | | | |
| 50 | 2,78 | 2,40 | 1,54 | 2,16 | 1,29 | 1,50 | 1,00 | 1,00 |
| 75 | 2,89 | 2,30 | 1,62 | 2,36 | 1,24 | 1,40 | 1,00 | 1,00 |
| 100 | 3,28 | 2,61 | 1,83 | 2,71 | 1,25 | 1,38 | 1,00 | 1,00 |
| 125 | 3,46 | 2,85 | 1,96 | 2,98 | 1,27 | 1,41 | 1,00 | 1,00 |
| 150 | 3,86 | 3,23 | 2,02 | 3,13 | 1,24 | 1,41 | 1,00 | 1,00 |
| 175 | 3,91 | 3,97 | 1,68 | 2,17 | 1,25 | 1,44 | 1,00 | 1,00 |
| 200 | 3,89 | 4,04 | 2,24 | 3,68 | 1,22 | 1,37 | 1,00 | 1,00 |

Finally, we set $T = T \cup (g,h) \sim (k,l)$ and the first iteration ends. The algorithm will execute five more iterations and will finally produce the optimal solution. In Fig. 6, we can see the optimal solutions produced by the exterior point algorithm, using all three initialization methods.

## 4. Computational experiments

An experimental computational study on randomly generated full dense TPs is presented in this section, to establish the practical value of the EPSAs. The results are very encouraging for this category of algorithms. All the compu-

Table 5
Normalized iterations and CPU time averages for TPs of class 2

| Size | PSA | | BFT | | SSF | | AKP | |
|---|---|---|---|---|---|---|---|---|
| | niter | cpu | niter | cpu | niter | cpu | niter | cpu |
| $n \times n$ | | | | | | | | |
| 50 | 2,93 | 2,23 | 1,75 | 2,04 | 1,57 | 1,63 | 1,00 | 1,00 |
| 75 | 3,34 | 2,51 | 1,80 | 2,30 | 1,48 | 1,61 | 1,00 | 1,00 |
| 100 | 3,53 | 2,71 | 1,98 | 2,70 | 1,45 | 1,61 | 1,00 | 1,00 |
| 125 | 4,10 | 3,41 | 2,22 | 3,15 | 1,50 | 1,74 | 1,00 | 1,00 |
| 150 | 4,12 | 3,33 | 2,20 | 3,07 | 1,46 | 1,63 | 1,00 | 1,00 |
| 175 | 4,55 | 3,81 | 2,53 | 3,78 | 1,48 | 1,73 | 1,00 | 1,00 |
| 200 | 4,58 | 4,65 | 2,41 | 3,56 | 1,46 | 1,70 | 1,00 | 1,00 |
| 225 | 4,53 | 3,81 | 2,51 | 3,77 | 1,43 | 1,66 | 1,00 | 1,00 |
| 250 | 4,91 | 5,10 | 2,65 | 4,08 | 1,48 | 1,78 | 1,00 | 1,00 |
| 275 | 5,04 | 5,25 | 2,91 | 4,53 | 1,43 | 1,71 | 1,00 | 1,00 |
| 300 | 4,92 | 5,06 | 2,73 | 4,23 | 1,43 | 1,70 | 1,00 | 1,00 |
| $n \times 2n$ | | | | | | | | |
| 50 | 3,33 | 2,85 | 1,93 | 2,56 | 1,72 | 1,96 | 1,00 | 1,00 |
| 75 | 3,71 | 3,62 | 2,21 | 3,24 | 1,63 | 1,96 | 1,00 | 1,00 |
| 100 | 4,34 | 4,41 | 2,38 | 3,64 | 1,73 | 2,17 | 1,00 | 1,00 |
| 125 | 4,81 | 5,17 | 2,72 | 4,37 | 1,70 | 2,21 | 1,00 | 1,00 |
| 150 | 4,96 | 4,93 | 2,89 | 4,48 | 1,68 | 2,06 | 1,00 | 1,00 |
| 175 | 5,01 | 5,03 | 2,76 | 4,45 | 1,65 | 2,08 | 1,00 | 1,00 |
| 200 | 5,18 | 4,51 | 2,91 | 4,63 | 1,63 | 2,07 | 1,00 | 1,00 |
| $2n \times n$ | | | | | | | | |
| 50 | 2,72 | 2,50 | 1,53 | 2,07 | 1,31 | 1,42 | 1,00 | 1,00 |
| 75 | 2,96 | 2,75 | 1,58 | 2,32 | 1,24 | 1,38 | 1,00 | 1,00 |
| 100 | 3,43 | 3,57 | 1,84 | 2,81 | 1,28 | 1,47 | 1,00 | 1,00 |
| 125 | 3,68 | 3,66 | 2,01 | 3,17 | 1,26 | 1,44 | 1,00 | 1,00 |
| 150 | 3,94 | 3,17 | 2,05 | 2,99 | 1,24 | 1,39 | 1,00 | 1,00 |
| 175 | 4,86 | 5,04 | 2,77 | 4,31 | 1,58 | 1,93 | 1,00 | 1,00 |
| 200 | 5,11 | 4,33 | 3,04 | 4,84 | 1,61 | 1,98 | 1,00 | 1,00 |

tations performed on a single processing node. The competitive algorithms have been programmed exactly the same way. The computer codes of the competitive algorithms were written in MATLAB 6.1, compiled and ran on a PC with 800EB MHz Pentium III processor and 512 Mb–133 MHz memory. The operating system was Windows 2000 Pro.

We experimented on randomly generated instances. We solved three classes of them. For each class we ran TPs of size $n \times n$, $n = 50, 75, \ldots, 300$, $n \times 2n$ and $2n \times n$ $(n = 50, 75, \ldots, 200)$. We solve 10 tests for each dimension. All problem classes were generated with the same set of random number seeds. Totally, we ran 750 randomly generated TPs. The classes we used are summarized as follows.

Table 6
Normalized iterations and CPU time averages for TPs of class 3

| Size | PSA | | BFT | | SSF | | AKP | |
|---|---|---|---|---|---|---|---|---|
| | niter | cpu | niter | cpu | niter | cpu | niter | cpu |
| $n \times n$ | | | | | | | | |
| 50 | 3,03 | 2,29 | 1,71 | 2,00 | 1,53 | 1,59 | 1,00 | 1,00 |
| 75 | 3,36 | 2,88 | 1,88 | 2,45 | 1,49 | 1,62 | 1,00 | 1,00 |
| 100 | 3,39 | 2,56 | 1,83 | 2,47 | 1,43 | 1,58 | 1,00 | 1,00 |
| 125 | 3,94 | 3,18 | 2,24 | 3,10 | 1,49 | 1,68 | 1,00 | 1,00 |
| 150 | 4,12 | 3,98 | 2,32 | 3,43 | 1,46 | 1,69 | 1,00 | 1,00 |
| 175 | 4,54 | 4,63 | 2,38 | 3,64 | 1,48 | 1,77 | 1,00 | 1,00 |
| 200 | 4,37 | 4,40 | 2,48 | 3,79 | 1,43 | 1,68 | 1,00 | 1,00 |
| 225 | 4,48 | 4,73 | 2,54 | 4,15 | 1,44 | 1,76 | 1,00 | 1,00 |
| 250 | 4,72 | 4,96 | 2,65 | 4,16 | 1,42 | 1,71 | 1,00 | 1,00 |
| 275 | 5,09 | 5,35 | 2,60 | 4,09 | 1,48 | 1,79 | 1,00 | 1,00 |
| 300 | 5,00 | 5,07 | 2,70 | 4,24 | 1,44 | 1,77 | 1,00 | 1,00 |
| $n \times 2n$ | | | | | | | | |
| 50 | 1,17 | 1,03 | 1,65 | 2,61 | 1,22 | 1,32 | 1,00 | 1,00 |
| 75 | 1,52 | 1,23 | 1,99 | 3,34 | 1,22 | 1,25 | 1,00 | 1,00 |
| 100 | 1,72 | 1,42 | 2,11 | 3,76 | 1,20 | 1,23 | 1,00 | 1,00 |
| 125 | 1,78 | 1,57 | 2,35 | 4,43 | 1,20 | 1,23 | 1,00 | 1,00 |
| 150 | 2,11 | 1,90 | 2,54 | 4,98 | 1,20 | 1,24 | 1,00 | 1,00 |
| 175 | 2,16 | 2,61 | 2,94 | 5,59 | 1,25 | 1,25 | 1,00 | 1,00 |
| 200 | 2,23 | 2,59 | 2,92 | 5,48 | 1,19 | 1,25 | 1,00 | 1,00 |
| $2n \times n$ | | | | | | | | |
| 50 | 1,37 | 1,19 | 1,33 | 1,95 | 1,47 | 1,74 | 1,00 | 1,00 |
| 75 | 1,67 | 1,23 | 1,50 | 2,42 | 1,44 | 1,71 | 1,00 | 1,00 |
| 100 | 1,97 | 1,96 | 1,54 | 2,68 | 1,43 | 1,76 | 1,00 | 1,00 |
| 125 | 2,16 | 2,24 | 1,59 | 2,88 | 1,40 | 1,73 | 1,00 | 1,00 |
| 150 | 2,43 | 2,58 | 1,59 | 2,97 | 1,40 | 1,75 | 1,00 | 1,00 |
| 175 | 2,68 | 2,68 | 1,60 | 3,00 | 1,39 | 1,77 | 1,00 | 1,00 |
| 200 | 3,08 | 2,71 | 1,68 | 3,03 | 1,51 | 1,78 | 1,00 | 1,00 |

Class 1: Asymmetric matrices of real numbers uniformly distributed in [1, $10^2$].

Class 2: Asymmetric matrices of real numbers uniformly distributed in [1, $10^6$].

Class 3: Asymmetric "heavy" $n \times n$ matrices of integer numbers. Each entry $c_{ij}$ is uniformly distributed in $[1, ij]$, $i, j = 1, 2, \ldots, n$.

Tables 1–3 show the statistics on the TPs used in our experimental computational study. Test problems are ordered by the increasing number of supply nodes. The following tables contain problem dimension, $n$, the average number of iterations, niter and CPU time to reach the 32-digit optimality, cpu. The reported running times are in CPU seconds. All running times are measured without any read time.

From Tables 1–3 we observe that the CPU time for all algorithms increases with respect to both $m$ and $n$. In general we could say that as $m$ increases and $m$ is greater than $n$, test problems take more CPU time to solve. For all examined dimensions AKP algorithm solved the test instances between $n$ and $4n$ iterations. This result consists a good practical performance. In order to show more clearly the superiority of EPSAs we present the following tables showing some balanced ratios relative with Tables 1–3. In Tables 4–6 we give the normalized iterations and CPU time averages for the corresponding classes. These ratios indicate how many times the AKP is faster over the other competitive algorithms. Our first conclusion is that AKP algorithm is winner implementation code in all test problems. For TPs of size $300\times300$ and for all classes, AKP report an average of 4.5 times faster than PSA in terms of number of iterations. For the same TPs, AKP report an average of 4.86 times faster in terms of CPU time.

Figs. 7 and 8 illustrate graphically the performance of AKP over the other algorithms. We plot the normalized ratios for TPs of size $n \times n$ taken from Table 6. In Fig. 7 we plot the normalized iterations averages, whereby in Fig. 8 the normalized CPU time averages. From the above experimental computational results we make the following observations. (i) The AKP algorithm is in general more efficient than the other competitive algorithms (PSA, BFT, SSF), (ii) One may infer a growth in the relative speed of AKP with respect to others as test problems sizes increase.

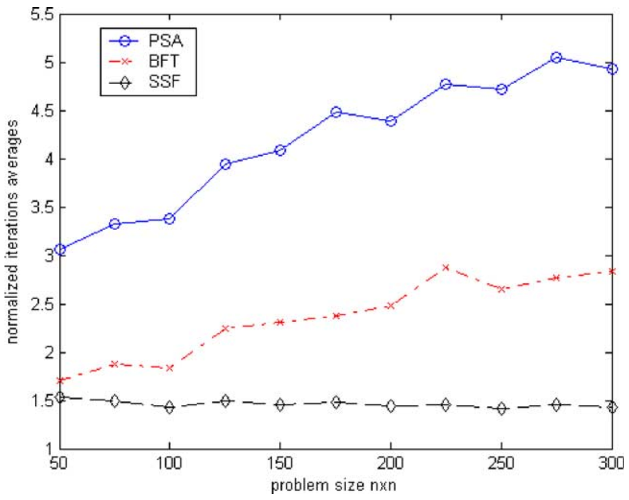

Fig. 7. Normalized iterations averages for TPs of class 3 and size $n \times n$.

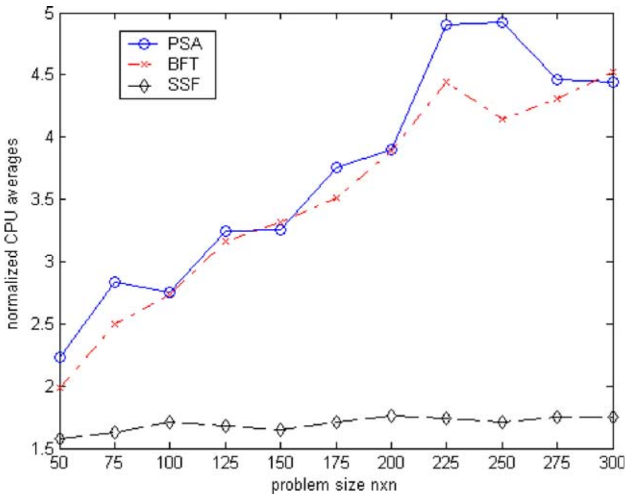

Fig. 8. Normalized CPU averages for TPs of class 3 and size $n \times n$.

## 5. Conclusions

In this paper an experimental computational study of EPSAs for the TP is presented. From the results of the previous section it is obvious that AKP algorithm is clearly the more efficient one. This algorithm has several advantages. It leads on the average to important reduction in the number of iterations and quite independently of the test problem. Our experimentation showed a precise ranking between the competitive algorithms. From the better to worst the rank is (i) AKP, (ii) SSF, (iii) BFT and (iv) PSA. This ranking raises the following question. Where does the computational power of the AKP come from? This power depends on the initial solution. The third initialization method, the AKP forest, seems to work very well in practice applied in TPs.

## References

[1] 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.

[2] 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, World Scientific Publishing Co., 2002, pp. 1–10.

[3] R. Ahuja, T. Magnanti, J. Orlin, Network Flows: Theory, Algorithms and Applications, Prentice-Hall, Englewood Clifs, NJ, 1993.

[4] H. Arsham, A.B. Khan, A simplex type algorithm for general transportation problems: an alternative to stepping-stone, J. Oper. Res. Soc. 40 (1989) 581–590.

[5] R.S. Barr, F. Glover, D. Klingman, The alternating basis algorithm for assignment problems, Math. Program. 13 (1977) 1–13.

[6] A. Charnes, W.W. Cooper, The stepping-stone method for explaining linear programming calculation in transportation problem, Manage. Sci. 1 (1954) 49–69.
[7] T.J. Cova, J.P. Johnson, A network flow model for lane-based evacuation routing, Transport. Res. Part A 37 (2003) 579–604.
[8] W.H. Cunningham, A network simplex method, Math. Program. 11 (1976) 105–116.
[9] G.B. Dantzig, Application of the simplex method to a transportation problem, in: T.C. Koopmans (Ed.), Activity of production and application, John Wiley & Sons, NY, 1951, pp. 359–373.
[10] T.C. Koopmans, Optimum utilization of the transportation system, Econometrica 17 (1949) 3–4.
[11] S. Liu, The total cost bounds of the transportation problem with varying demand and supply, OMEGA 31 (2003) 247–251.
[12] J.B. Orlin, S.A. Plotkin, E. Tardos, Polynomial dual network simplex algorithms, Math. Program. 60 (1993) 255–276.
[13] C. Papamanthou, K. Paparrizos, N. Samaras, A comparative computational study of exterior point algorithms for the assignment problem, J. Algor., submitted for publication.
[14] K. Paparrizos, A network exterior point algorithm, Presented at the EURO X Conference on Operational Research, Beograd, Yugoslavia, 1989.
[15] K. Paparrizos, A simplex type algorithm for assignment problems initialized with a solution that is neither primal nor dual feasible, in: Proceedings of XXI Conference on Applications of Mathematics in Industry and Business, Varna, Bulgaria, 1995, pp. 152–159.
[16] K. Paparrizos, An exterior point simplex algorithm for general linear problems, Ann. Oper. Res. 32 (1993) 497–508.
[17] K. Paparrizos, An infeasible (exterior point) simplex algorithm for assignment problems, Math. Program. 51 (1991) 45–54.
[18] L. Portugal, F. Bastos, J. Judice, J. Paixao, T. Terlaky, An investigation of interior point algorithms for the linear transportation problem, SIAM J. Sci. Comput. 17 (1996) 1202–1223.
[19] R. Ruiz, C. Maroto, J. Alcaraz, A decision support system for a real vehicle routing problem, Euro. J. Oper. Res. 153 (2003) 593–606.
[20] W.L. Winston, Operations Research, Applications and Algorithms, third ed., Duxbury Press, 1997.
[21] C. Zhang, J. Liu, Y. Wan, K.G. Murty, R.J. Linn, Storage space allocation in container terminals, Transport. Res. Part B 37 (2003) 883–903.