DEPARTMENT OF OPERATIONS RESEARCH
UNIVERSITY OF AARHUS

Working Paper no. 2005 / 2

2005 / 12 / 19

A note on ranking assignments using reoptimization

Christian Roed Pedersen, Lars Relund Nielsen
and Kim Allan Andersen

ISSN 1600-8987

Department of Mathematical Sciences
Telephone: +45 8942 1111
E-mail: institut@imf.au.dk

Building 530, Ny Munkegade
DK-8000 Aarhus C, Denmark
URL: www.imf.au.dk

A note on ranking assignments using reoptimization

Christian Roed Pedersen*
Department of Operations Research
University of Aarhus
Ny Munkegade, Building 1530
8000 Aarhus C
Denmark

Lars Relund Nielsen Kim Allan Andersen
Department of Accounting, Finance and Logistics
Aarhus School of Business
Fuglesangs Allé 4
DK-8210 Aarhus V
Denmark

December 19, 2005

Abstract

We consider the problem of ranking assignments according to cost in the classical linear assignment problem. An algorithm partitioning the set of possible assignments, as suggested by Murty, is presented where, for each partition, the optimal assignment is calculated using a new reoptimization technique. Computational results for the new algorithm are presented.

Keywords: Linear Assignment Problem, Ranking, K best solutions.

1 Introduction

The *linear assignment problem* (AP) is a well-known problem and may be considered as the problem of assigning n workers to n jobs. Each worker must be assigned to exactly one job. The objective is to minimize total cost.

In an annotated bibliography authored by Dell'Amico and Martello [4], more than 100 papers on the problem are mentioned. Kuhn [9, 10] suggested the first polynomial method for the solution of AP, called the *Hungarian method* with $\mathcal{O}(n^4)$

*Corresponding author, e-mail: roed@imf.au.dk.

complexity. Since 1955 several other algorithms for AP have been developed. Some of the most efficient algorithms are the class of *successive shortest path algorithms*[1] with an $\mathcal{O}(n^3)$ complexity, see e.g. Jonker and Volgenant [8], Tomizawa [17]. An excellent survey is given by Dell'Amico and Toth [5] including comparative tests of several implementations of different AP algorithms.

As for other optimization problems, the assignment problem can be generalized to *ranking the first K assignments* in nondecreasing order of cost. Applications of ranking assignments are numerous. First, ranking assignments can be used to generate near optimal solutions which may be better from a practical point of view when presented to the decision maker. Second, applications of AP may include constraints which are hard to specify formally or hard to optimize. Here an optimal assignment can be found by enumerating suboptimal assignments until an assignment satisfying the hard constraints is found. Last but not least, ranking assignments appear as a subproblem within algorithms for solving the *bicriterion assignment problem* and related extensions, see for example Pedersen, Nielsen, and Andersen [15].

Several algorithms for ranking assignments have been suggested. They may be classified using two identifiers: A specific *branching technique* is used to partition the set of possible assignments into smaller subsets, and a *solution technique* needed for ranking the assignments is used to find an assignment for each subset.

Murty [12] suggested a branching technique where the set of possible assignments is partitioned into at most $n-1$ disjoint subsets for each additional ranking made. The Hungarian algorithm was used to find the best assignment for each subset resulting in an $\mathcal{O}(Kn^5)$ complexity. However, applying e.g. a successive shortest path algorithm may improve the overall complexity to $\mathcal{O}(Kn^4)$. Later, the branching technique in [12] was applied in the more general framework of finding the K best solutions to a discrete optimization problem by Lawler [11].

Hamacher and Queyranne [7] presented an alternative general framework for ranking solutions of combinatorial problems, later specialized for bipartite matchings by Chegireddy and Hamacher [3]. There, an alternative branching technique is suggested, partitioning the set into at most two subsets for each additional ranking. For each subset, the second best assignment has to be calculated. Different solution techniques are suggested. One consists of identifying the second best assignment by a shortest cycle determination in an auxiliary network. The shortest cycle can be found by solving at most n shortest paths problems resulting in an overall $\mathcal{O}(Kn^3)$ time complexity.

Recently, Pascoal, Captivo, and Clímaco [14] presented a ranking algorithm with the same branching technique as in [12]. However, by considering the subsets in reverse order when applying their solution technique, they are able to reoptimize the solution from the previous subset considered and find the best assignment by solving a single shortest path problem yielding the same running time as in [3].

The solution techniques in all the above ranking algorithms use shortest path methods to find the best assignment for each subset. Methods based on shortest paths are *dual algorithms*. Dual feasibility exists and primal feasibility has to be reached. Tomizawa [17] noted that the original costs in the assignment may be replaced with the reduced costs when using successive shortest path algorithms.

[1] Also known as shortest augmenting paths algorithms.

Since the reduced costs are non-negative, the shortest path may be found using the algorithm of Dijkstra [6].

In spite of the connection between the dual variables and successive shortest path algorithms no one has considered updating the dual variables of the previous solution before the shortest path algorithm is applied to a subset. In this paper, we present an algorithm for ranking assignments, using the branching technique of Murty [12]. For each subset, a solution technique is used where – by updating the dual variables – only a single shortest path problem has to be solved. Hence, the overall time complexity of the proposed method is $\mathcal{O}(Kn^3)$.

After a short overview over the dual properties in AP and successive shortest path algorithms in Section 2, we present the new ranking algorithm in Section 3. In Section 4, computational results are given, and conclusions are drawn in Section 5.

2 Preliminaries

Let $G = (U \cup V, E)$ be a bipartite network with node sets $U = V = \{1, \ldots, n\}$ and edge set $E = \{(i, j) : i \in U, j \in V\}$ with cost c_{ij} on each edge (i, j). Note that non-existing edges can be represented as arcs having infinity cost. The *assignment problem* consists in assigning – with minimum total cost – each node in U to a node in V.

Due to its totally unimodular constraint matrix AP can be formulated as the continuous linear program

$$\begin{aligned} \min \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & \\ \text{st.} \quad & \sum_{j=1}^{n} x_{ij} = 1, & i = 1, \ldots, n \\ & \sum_{i=1}^{n} x_{ij} = 1, & j = 1, \ldots, n \\ & x_{ij} \geq 0 & i, j = 1, \ldots, n \end{aligned} \tag{1}$$

In an optimal solution to (1), $x_{ij} = 1$ if node i is assigned node j, and zero otherwise. A feasible solution x to (1) is called an *assignment*. Using the network formulation, an assignment may alternatively be written as $a = \{(1, j_1), \ldots, (n, j_n)\}$ where $(i, j) \in a$ if and only if $x_{ij} = 1$. A *partial primal solution* is a solution in which less than n variables is assigned value one, and the constraints in (1) are satisfied with a $\leq$ sign instead of equality. Note that a partial primal solution corresponds to a *partial assignment* $a = \{(i_1, j_1), \ldots, (i_q, j_q)\}$.

By associating *dual variables* u_i and v_j with the constraints above, the corresponding dual problem is

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j & \\ \text{st.} \quad & u_i + v_j \leq c_{ij}, & i, j = 1, \ldots, n \end{aligned} \tag{2}$$

```
1 procedure SuccSP()
2    (a, u, v) :=Preprocess([c_ij]);
3    while (|a| < n) do
4       G' :=BuildAuxNetwork(a, u, v);
5       p :=FindAugmentPath(G');
6       a :=AugmentSolution(p);
7       (v, u) :=AdjustDualSolution(p);
8     end while
9 end procedure
```

Figure 1: The successive shortest path algorithm

Given the *reduced costs* $\bar{c}_{ij} = c_{ij} - u_i - v_j$ of AP, the *complementary slackness* conditions become

$$x_{ij}\bar{c}_{ij} = 0,\ i, j = 1, \ldots, n \tag{3}$$

Successive shortest path algorithms for AP are dual methods. Dual feasibility exists and the optimal solution is built step-by-step by iteratively adding assignments to a current partial primal solution.

A successive shortest path algorithm consists of two phases. In phase one, the cost matrix $[c_{ij}]$ is preprocessed and a partial primal solution (or partial assignment) and a dual feasible solution are determined which satisfy the complementary slackness conditions (3). In phase two, the partial primal solution is augmented by adding one row/column assignment at a time, until the solution becomes feasible. At each step in phase two, the dual solution is updated so that complementary slackness still holds. Hence, at the end of the second phase, the current primal and dual solutions are optimal.

A pseudo-code for the successive shortest path algorithm is given in Figure 1. Phase one is executed by function `Preprocess` which returns a partial assignment a and a dual feasible solution (u, v) satisfying (3). Phase two is executed on lines 3–8.

If $|a| \neq n$ then all nodes in U have not been assigned to a node in V and function `BuildAuxNetwork` builds an auxiliary directed network G' constructed from G. In the following, let i denote a node in U and j a node in V. $G' = (V', A')$ is defined as follows:

$$\begin{aligned}
&V' = \{s\} \cup U \cup V, \quad A' = A'_s \cup A'_d \cup A'_r \\
&A'_r = \{(j, i) : (i, j) \in a\} \\
&A'_d = \{(i, j) : (i, j) \notin a\} \\
&A'_s = \{(s, i) : (i, j) \notin a,\ \forall j = 1, \ldots, n\}
\end{aligned}$$

The set A'_d is the set of *directed* arcs, A'_r is the set of *reverse* arcs and A'_s is the set of *auxiliary* arcs. The arcs in A'_s are assigned a zero cost. Each arc (i, j) in A'_d is assigned reduced cost $\bar{c}_{ij}$ and each arc (j, i) in A'_r is assigned cost $-\bar{c}_{ij} = 0$ due to (3).

It is easy to see that any directed path in G' starting from s contains an arc in A'_d and an arc in A'_r, alternatingly. Such paths are called *alternating paths*. If

the directed path p terminates with an unassigned node in V, then it is called an *augmenting path*. Furthermore, the following lemma is well-known.

Lemma 1. *Given partial assignment a, network G' and an augmenting path p in G', set*

$$\bar{a} = (p \cap A'_d) \cup (a \setminus (p \cap A'_r)) \tag{4}$$

Then $|\bar{a}| = |a| + 1$.

That is, by removing assignments in a corresponding to the reverse arcs in p and adding the directed arcs in p to a, the number of assignments in the (partial) assignment $\bar{a}$ has increased by one. Hence, finding the shortest augmenting path in G' is equivalent to finding a minimum cost (partial) assignment $\bar{a}$ with $|a| + 1$ assignments. As a result, AP can be solved by identifying at most n successive shortest augmenting paths. Since the reduced costs are non-negative, each path can be determined through Dijkstra's algorithm running in $\mathcal{O}(n^2)$ time. Therefore, the overall computational complexity of a successive shortest path algorithm is $\mathcal{O}(n^3)$.

In procedure `SuccSP` the shortest augmenting path p is found using function `FindAugmentPath` and next the (partial) assignment a is updated as in (4) using function `AugmentSolution`. Finally, the dual variables are updated using function `AdjustDualSolution` such that (3) holds.

For an efficient implementation of procedure `succSP` see for instance Jonker and Volgenant [8]. Here extensive preprocessing is used in `Preprocess` and network G' is maintained implicitly in the data structures. Function `FindAugmentPath` calculates the augmented path using a specialized version of Dijkstra and the new solution is found in `AugmentSolution` by traversing the path. Finally, the dual solution is adjusted by traversing the path. For further details see [8].

3 Ranking assignments

We consider the problem of ranking the first K assignments in nondecreasing order of cost, i.e. finding the K best assignments $a_1, \ldots, a_K$ satisfying

1. $c(a_i) \leq c(a_{i+1}),\ i = 1, \ldots, K.$
2. $c(a_K) \leq c(a),\ \forall a \notin \{a_1, \ldots, a_K\}.$

where $c(a)$ denotes the cost of assignment a.

An algorithm for ranking assignments in general uses a specific *branching technique* to partition the set of possible assignments into smaller subsets and uses a *solution technique* to find an assignment for each subset needed for ranking the assignments. Let $\mathcal{A}$ denote the set of possible assignments. In this paper we use the branching technique described in [12] where the set $\mathcal{A}$ is partitioned into smaller subsets as follows. Given the optimal assignment $a_1 = \{(1, j_1), \ldots, (n, j_n)\}$, the set $\mathcal{A} \setminus \{a_1\}$ is partitioned into $n - 1$ disjoint subsets $\mathcal{A}^i$, $i = 1, \ldots, n-1$, where

$$\mathcal{A}^i = \{a : \{(1, j_1), \ldots, (i-1, j_{i-1})\} \in a, (i, j_i) \notin a\},\ \ i = 1, \ldots, n-1.$$

```
1  procedure K-AP()
2     a :=SuccSP();
3     Φ := Φ ∪ {(a, A)};
4     for (k := 1 to K) do
5        (â, Â) := arg min{c(ā) : (ā, Ā) ∈ Φ};
6        if ((â, Â) =null) then STOP; else OUTPUT a_k := â;
7        Φ := Φ \ {(â, Â)};
8        for (i:=1 to n-1) do
9           â^i :=FindOptimal(Â^i);
10          if (c(â^i) < ∞) then Φ := Φ ∪ {(â^i, Â^i)};
11       end for
12    end for
13 end procedure
```

Figure 2: The ranking assignments algorithm.

We say that $\{(1, j_1), \ldots, (i-1, j_{i-1})\}$ is *forced* to be in all assignments belonging to $\mathcal{A}^i$. Clearly, the second best assignment a_2 can be found by using a solution technique to find the optimal assignment in the sets $\mathcal{A}^i$, $i = 1, \ldots, n-1$. Moreover, the branching technique can be applied recursively to subsets $\mathcal{A}^i \subset \mathcal{A}$.

The pseudo code for the ranking algorithm, named `K-AP`, is shown in Figure 2. The algorithm implicitly maintains a candidate set Φ of pairs $(\bar{a}, \bar{\mathcal{A}})$, where $\bar{a}$ is the optimal assignment in (sub)set $\bar{\mathcal{A}}$. Assuming that the first $k-1$ assignments $a_1, \ldots, a_{k-1}$ have been found, the current candidate set represents the partition of $\mathcal{A} \setminus \{a_1, \ldots, a_{k-1}\}$. Assignment a_k is then found by selecting and removing the pair $(\hat{a}, \hat{\mathcal{A}})$ containing the assignment with minimum cost in the candidate set (lines 5–7). Next, the branching technique is used to partition $\hat{\mathcal{A}}$, possibly obtaining new pairs that are added to the candidate set (lines 8–11). Note that, in practice, we do not have to consider all subsets in the partition of $\hat{\mathcal{A}}$. Consider the case where (i, j_i) was forced to be contained in an assignment belonging to $\hat{\mathcal{A}}$ in some previous partition. Hence $\hat{\mathcal{A}}^i = \varnothing$, since (i, j_i) must not be contained in an assignment in $\hat{\mathcal{A}}^i$. In this case, we may assume that $\hat{\mathcal{A}}^i$ is not generated by the algorithm.

Function `FindOptimal` represents the solution technique applied for finding the optimal assignment in the given subset. Consider that partition $\hat{\mathcal{A}}$ has optimal assignment $\hat{a} = \{(1, j_1), \ldots, (n, j_n)\}$ and assume that all assignments in $\hat{\mathcal{A}}$ cannot contain $(l_1, t_1), \ldots, (l_q, t_q)$ (previous partitions). Recall that $(1, j_1), \ldots, (i-1, j_{i-1})$ are forced to be in all assignments belonging to subset $\hat{\mathcal{A}}^i$ ($\subseteq \hat{\mathcal{A}}$). Therefore, assuming $\hat{\mathcal{A}}^i$ is nonempty, the optimal assignment can be found solving an AP of size $n - (i-1)$ where

1. Rows $\{1, \ldots, i-1\}$ and columns $\{j_1, \ldots, j_{i-1}\}$ have been removed from the reduced cost matrix $[\bar{c}_{ij}]$, i.e. we do not consider these indexes in (1) and (2).

2. The reduced cost in cells (i, j_i) and $(l_1, t_1), \ldots, (l_q, t_q)$ is set to infinity.

Given a nonempty subset $\hat{\mathcal{A}}^i$, let $AP(\hat{\mathcal{A}}^i)$ denote the AP defined as above by subset $\hat{\mathcal{A}}^i$. If the successive shortest path algorithm, `SuccSP`, is used as the solution

technique to find the optimal assignment to $AP(\hat{\mathcal{A}}^i)$, $i = 1, \ldots, n-1$, the overall complexity of `K-AP` is $\mathcal{O}(Kn^4)$. However, the optimal assignment to $AP(\hat{\mathcal{A}}^i)$ can be found using reoptimization – thereby reducing the complexity of the algorithm.

Lemma 2. *Let $\hat{a}$ denote the optimal assignment in subset $\hat{\mathcal{A}}$ found by solving $AP(\hat{\mathcal{A}})$ and let $(\hat{u}_i, \hat{v}_j)$ denote the corresponding dual values. Define*

$$a\,(i) = \hat{a} \setminus \{(i, j_i)\} \tag{5}$$

and dual variables

$$\begin{aligned} u_i &= \hat{u}_i + \min\nolimits_{j \in \{1,\ldots,n\} \setminus \{j_1,\ldots,j_i\}} \{c_{ij} - \hat{u}_i - \hat{v}_j\} \\ u_r &= \hat{u}_r,\ r \in \{i+1, \ldots, n\} \\ v_{j_i} &= \hat{v}_{j_i} + \min\nolimits_{r \in \{i+1,\ldots,n\}} \{c_{rj_i} - \hat{u}_r - \hat{v}_{j_i}\} \\ v_j &= \hat{v}_j,\ j \in \{1, \ldots, n\} \setminus \{j_1, \ldots, j_i\} \end{aligned} \tag{6}$$

Then $a(i)$ is a partial assignment of size $n-1$ and (u, v) is a dual feasible solution to $AP(\hat{\mathcal{A}}^i)$, which satisfy the complementary slackness conditions (3).

Proof. Given $(\hat{u}, \hat{v})$, let $\hat{c}$ denote the corresponding non-negative reduced costs. Since $u_r = \hat{u}_r$, $r \in \{i+1, \ldots, n\}$, and $v_j = \hat{v}_j$, $j \in \{1, \ldots, n\} \setminus \{j_1, \ldots, j_i\}$, the reduced cost matrix to $AP(\hat{\mathcal{A}}^i)$ using (u, v) from (6) becomes

	j_i	j_{i+1}	$\cdots$	j_n
i	∞	$c_{ij_{i+1}} - u_i - \hat{v}_{j_{i+1}}$	$\cdots$	$c_{ij_n} - u_i - \hat{v}_{j_n}$
$i+1$	$c_{i+1j_i} - \hat{u}_{i+1} - v_{j_i}$	$\hat{c}_{i+1j_{i+1}}$	$\cdots$	$\hat{c}_{i+1j_n}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
n	$c_{nj_i} - \hat{u}_n - v_{j_i}$	$\hat{c}_{nj_{i+1}}$	$\cdots$	$\hat{c}_{nj_n}$

That is, only the reduced costs in column j_i and row i have changed. Due to (6) the reduced costs in row i satisfy

$$c_{ij} - u_i - v_j \geq c_{ij} - (\hat{u}_i + (c_{ij} - \hat{u}_i - \hat{v}_j)) - \hat{v}_j = 0,\ \ \forall j \in \{1, \ldots, n\} \setminus \{j_1, \ldots, j_i\}$$

Similar results hold for the reduced costs in column j_i. Hence (u, v) is a dual feasible solution to $AP(\hat{\mathcal{A}}^i)$. Moreover, since the reduced costs corresponding to the elements in assignment $a\,(i)$ have not changed, the complementary slackness conditions (3) still holds. □

Using Lemma 2, the optimal assignment in $\hat{\mathcal{A}}^i$ can be found as follows. Simply consider the partial assignment (5), update the dual variables according to (6) and find the shortest augmenting path in the corresponding auxiliary network. Furthermore, due to Lemma 1 and the fact that the length of the partial assignment (5) is $n-1$, we have the following result.

Theorem 1. *Using partial assignment* (5) *and dual values* (6)*, the optimal assignment in subset $\hat{\mathcal{A}}^i$ can be found by solving a single shortest path problem.*

```
1 procedure FindOptimal(Â^i)
2     a :=CreatePartial();
3     (u, v) :=ModifyDual();
4     G' :=BuildAuxNetwork(a, u, v);
5     p :=FindAugmentPath(G');
6     a :=AugmentSolution(p);
7 end procedure
```

Figure 3: Finding optimal solution for a subset.

Note that, Theorem 1 does not hold if the Hungarian algorithm was used instead. In this case examples can be constructed where more than one iteration is needed.

A pseudo-code for the reoptimization algorithm is given in Figure 3. Here first the partial assignment (5) and the dual values (6) are calculated. Next, the auxiliary network is built. Finally, the shortest augmenting path and the corresponding solution are found.

Using Dijkstra to find the shortest path function `FindOptimal`, runs in $\mathcal{O}(n^2)$ and hence we obtain the overall complexity of `K-AP`.

Theorem 2. *The K best assignments using procedure `K-AP` can be found in $\mathcal{O}(Kn^3)$ time.*

4 Computational results

To validate the effectiveness of the reoptimization based solution procedure, we implemented two versions of the ranking assignment algorithm `K-AP`.

The first algorithm, `K-AP_SuccSP`, is a straightforward implementation where the successive shortest path algorithm, `SuccSP`, is used as function `FindOptimal` (line 9 in `K-AP`) to find the optimal assignment in each subset $\mathcal{A}^i$ (or to problem $AP(\hat{\mathcal{A}}^i)$). For each problem $AP(\hat{\mathcal{A}}^i)$, we use the successive shortest path algorithm implementation of Jonker and Volgenant [8] slightly modified, though, such that we can solve problems of varying size.

The second algorithm, `K-AP_Reopt`, uses the reoptimization solution technique given in Figure 3. Here `FindAugmentPath` finds the shortest augmenting path using the implementation of Jonker and Volgenant [8].

In both algorithms, the candidate set Φ of pairs $(\bar{a}, \bar{\mathcal{A}})$ is maintained implicitly using a binary tree as described in Nielsen [13, p137], and a 4-heap is used to sort the costs in nondecreasing order, see Tarjan [16].

The algorithms were implemented in C++ and compiled with the GNU C++ compiler using optimize option `-O`. Moreover, all computations were performed on an Intel Xeon 2.67 GHz computer with 6 GB RAM using operating system Red Hat Enterprise Linux version 4.0.

To yield consistency in literature, we compared the two algorithms on the instances provided in Pascoal et al. [14] plus a few larger instances not reported upon in that paper. We consider two separate groups of instances, where, only for the

size	ave CPU K-AP_SuccSP	K-AP_Reopt	ratio
50	0.30	0.07	4.54
100	6.64	0.77	8.68
150	30.69	2.97	10.34
200	87.21	7.60	11.48
250	176.19	14.33	12.30
300	299.95	23.06	13.01

Table 1: Average CPU times for K=100, Pascoal et al. [14] instances.

second group, repetitions are made for a given problem size. Therefore, the main part of the results presented in this section are on the second group of test instances.

The first group of test instances consists of assignment problems on complete bipartite networks with $n \in \{100, 200, \ldots, 800\}$ and cost randomly drawn in $\{1, \ldots, 100\}$, taken from the OR-library[2] and first used in Beasley [1]. Only one instance of each problem size is solved, and the $K = 100$ best assignments are ranked. In Figure 4 we display the CPU time (in seconds) against K for the individual problem sizes. For all instances, the reoptimization algorithm shows significant superiority to the algorithm without reoptimization. Furthermore, for `K-AP_Reopt`, the CPU time seems to be linear dependent on the number of assignments to rank.

For the second group of test instances, the focus is on ranking the $K = 100$ best assignments for somewhat smaller complete bipartite networks of size $n \in \{50, 100, \ldots, 300\}$ and cost randomly drawn in $\{0, \ldots, 1000\}$. For each problem size, ten instances generated with different seeds were solved. These test problems were kindly provided to us by the authors Pascoal et al. [14]. Figure 5 shows the results for these problem classes and shows average CPU time to be linear dependent on K for both algorithms. Again, the superiority of `K-AP_Reopt` is confirmed by Figure 5 and can also be seen by numerical comparison in Table 1. Here, for each possible problem size, we display the average CPU times for ranking the 100 best assignments for both algorithms and the ratio between these CPU times. Since this ratio is increasing in problem size, it shows that the reoptimization routine becomes increasingly more valuable as the problem size increases.

For the second group of instances, we also plot the CPU times against size for two specific K-values ($K = 50$ and $K = 100$), (see Figure 6). Both algorithms show more than a linear growth in CPU time with increasing size. However, it is less than exponential growth, as can be seen in Figure 7.

[2] `http://people.brunel.ac.uk/~mastjjb/jeb/info.html` (see also [2]).

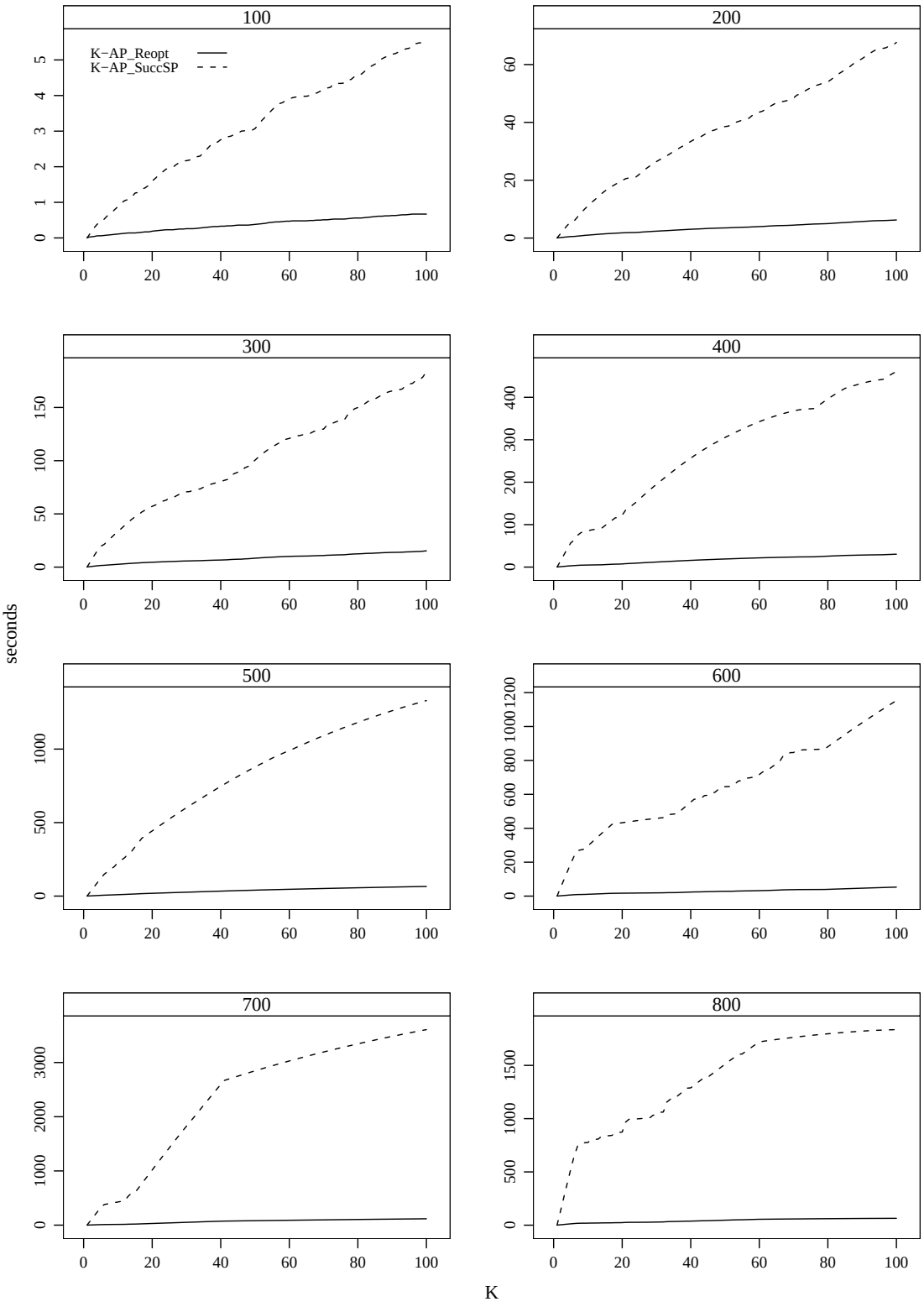

Figure 4: CPU time against K for Beasley [1] instances $n = 100, \ldots, 800$.

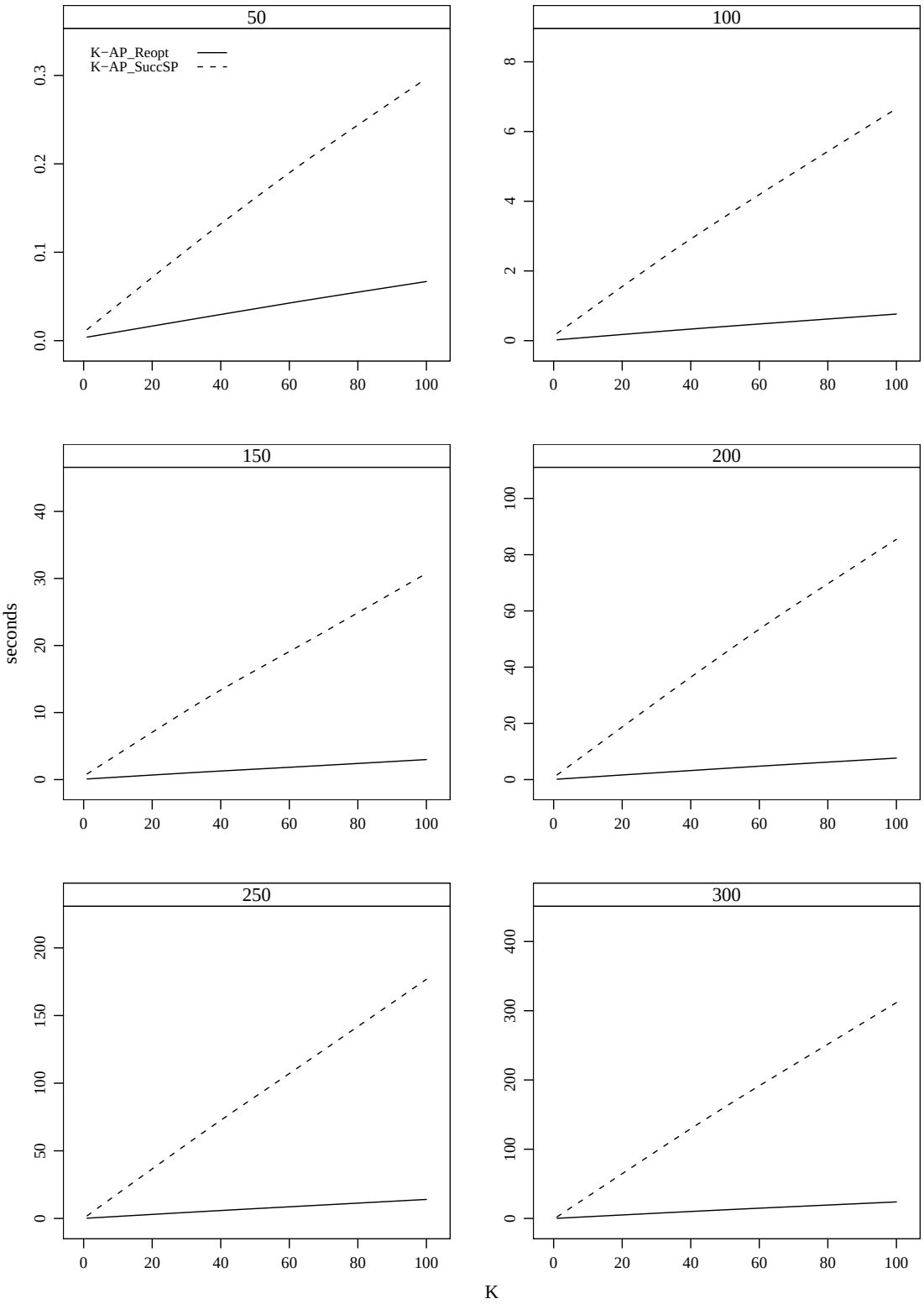

Figure 5: Average CPU time against K for Pascoal et al. [14] instances $n = 50, \ldots, 300$.

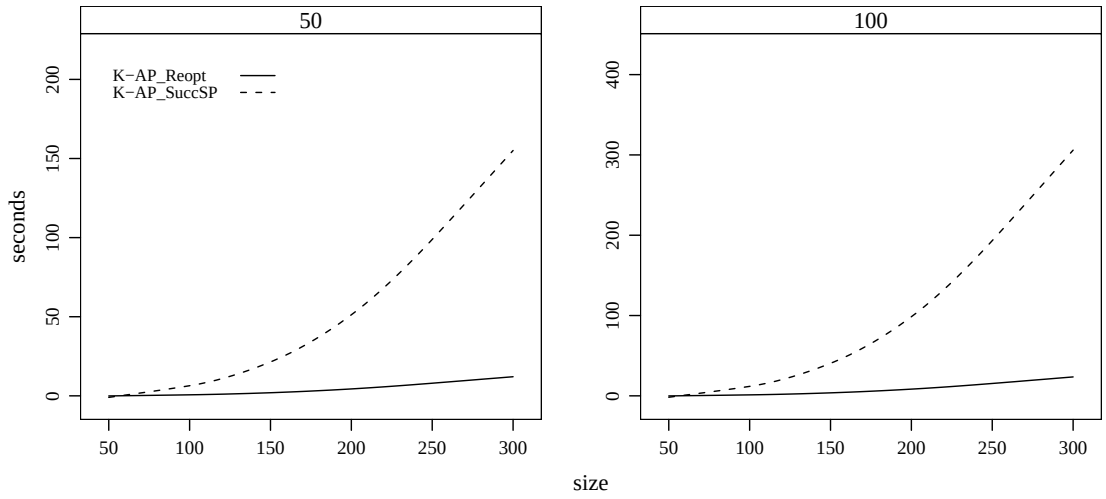

Figure 6: Average CPU time against size for Pascoal et al. [14] instances, $K = 50$ and 100.

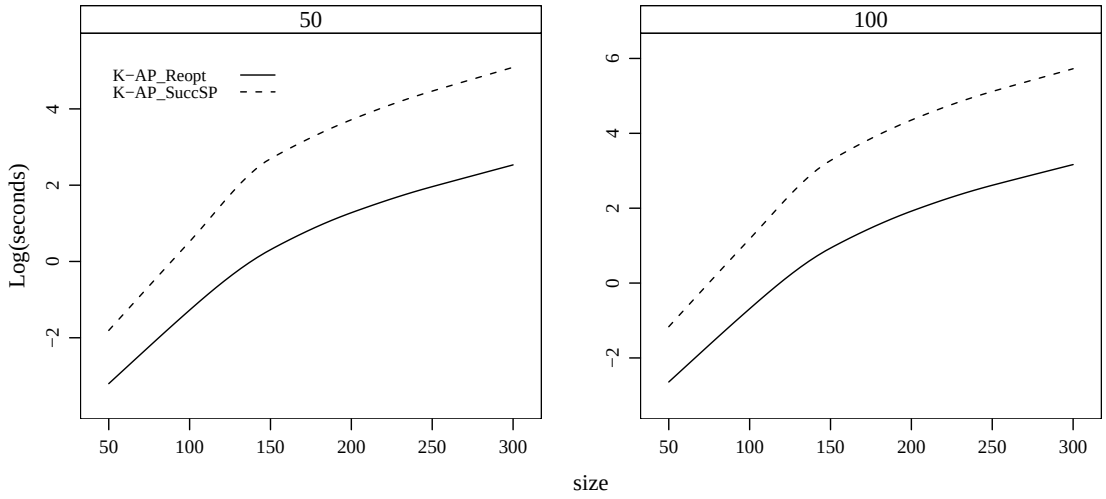

Figure 7: Logarithm of average CPU time against size for Pascoal et al. [14] instances, $K = 50$ and 100.

5 Conclusion

In this paper we have presented a new algorithm for ranking assignments using a branching technique of Murty [12]. The algorithm utilizes reoptimization based on easy updating of the dual variables in a successive shortest path procedure. To validate the effectiveness of the new algorithm, we compared it to a straightforward implementation where the successive shortest path algorithm is used to find the optimal assignment for each subset. Numerical tests on instances previously used in literature were presented.

The new algorithm was shown to have the best known theoretical time com-

plexity and the numerical tests showed that in practice it performs better than the straightforward implementation. Furthermore, it was concluded that reoptimization becomes exceedingly vital when the considered problem size increases.

References

[1] J.E. Beasley. Linear programming on cray supercomputers. *Journal of the Operations Research Society*, 41(2):133–139, 1990.

[2] J.E. Beasley. Or-library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(2):1069–1072, 1990.

[3] C.R. Chegireddy and H.W. Hamacher. Algorithms for finding k-best perfect matchings. *Discrete Applied Mathematics*, 18:155–165, 1987.

[4] M. Dell'Amico and S. Martello. Linear assignment. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, pages 355–371. Wiley, Chichester, 1997.

[5] M. Dell'Amico and P. Toth. Algorithms and codes for dense assignment problems: the state of the art. *Discrete Appl. Math.*, 100(1-2):17–48, 2000. doi: 10.1016/S0166-218X(99)00172-9.

[6] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[7] H.W. Hamacher and M. Queyranne. k-best solutions to combinatorial optimization problems. *Annals of Operations Research*, 6(4):123–143, 1985.

[8] R. Jonker and A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38:325–340, 1987.

[9] H.W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quart.*, 2:83–97, 1955.

[10] H.W. Kuhn. Variants of the hungarian method for the assignment problem. *Naval Research Logistics Quart.*, 3:253–258, 1956.

[11] E.L. Lawler. A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path. *Management Science*, 18(7):401–405, March 1972.

[12] K.G Murty. An algorithm for ranking all the assignments in order of increasing cost. *Operations Research*, 16:682–687, 1968.

[13] L.R. Nielsen. *Route Choice in Stochastic Time-Dependent Networks*. PhD thesis, Department of Operations Research, University of Aarhus, 2004.

[14] M. Pascoal, M. Eugénia Captivo, and J. Clímaco. A note on a new variant of Murty's ranking assignments algorithm. *4OR: Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1(3):243–255, 2003. doi: 10.1007/s10288-003-0021-7.

[15] C.R. Pedersen, L.R. Nielsen, and K.A. Andersen. On the bicriterion multi modal assignment problem. Working Paper No. 2005/3, Department of Operations Research, University of Aarhus, 2005.

[16] R.E. Tarjan. *Data Structures and Network Algorithms*, volume 44 of *CBMS-NSF Conference Series*. SIAM, 1983.

[17] N. Tomizawa. On some techniques useful for the solution of transportation problems. *Networks*, 1:173–194, 1971.