4OR 1: 243–255 (2003)

DOI: 10.1007/s10288-003-0021-7

A note on a new variant of Murty's ranking assignments algorithm

Marta Pascoal[1,2]*, **M. Eugénia Captivo**[3], **and João Clímaco**[4,5]

[1] Centro de Informática e Sistemas
[2] Departamento de Matemática, Universidade de Coimbra, Apartado 3008, 3001-454 Coimbra, Portugal (e-mail: marta@mat.uc.pt)
[3] DEIO-CIO, Faculdade de Ciências, Universidade de Lisboa, Campo Grande, Bloco C2, 1749-016 Lisboa, Portugal (e-mail: mecaptivo@fc.ul.pt)
[4] Instituto de Engenharia de Sistemas e Computadores – Coimbra, Rua Antero de Quental, 199, 3000-033 Coimbra, Portugal (e-mail: jclimaco@inescc.pt)
[5] Faculdade de Economia da Universidade de Coimbra, Avenida Dias da Silva, 165, 3004-512 Coimbra, Portugal

Received: August 2002 / Revised version: March 2003

Abstract. In this paper a variant of Murty's algorithm for ranking assignments according to cost is presented. It is shown that the worst-case computational complexity is better in this variant than in the original form of the algorithm. Computational results comparing three methods for ranking assignments are reported. They show that the behaviour of the new variant is also better in practice.

Key words: Graph theory, mathematical programming, combinatorics

AMS classification: 90C27, 05C85

1 Introduction

The assignment problem, is a well-known combinatorial problem, introduced in 1955 by Kuhn [10], who also proposed an algorithm to solve it. Since 1955 several other algorithms for the assignment problem have been developed. A survey including comparative tests was presented in [7].

As for other optimisation problems, the assignment problem can be generalized to enumerate the first K assignments, for some $K \in \mathbb{N}$, that is, the K least cost assignments, defining the so called ranking assignments problem. Listing solutions

* The research of Marta Pascoal was developed within CISUC and partially supported by the Portuguese Ministry of Science and Technology (MCT), under PRAXIS XXI Project of JNICT.

by order of cost is, for instance, used in the generation of alternative solutions, in the resolution of constrained assignment problems and in multicriteria analysis. Since, in general, several solutions have to be ranked, the efficiency of ranking algorithms is extremely important. There are only two algorithms in the literature for ranking assignments. The first was proposed by Murty in 1968 [12], while in 1985/86 Hamacher and Queyranne suggested an alternative general algorithm for ranking solutions of combinatorial problems [9], later specialized for bipartite matchings by Chegireddy and Hamacher [6].

The purpose of this work is to present a variant of Murty's algorithm, based on the use of a specific order for analysing each k-th assignment, instead of implementing Murty's algorithm straightforwardly. Although appearing to be a simple modification, it ends up in replacing the resolution of assignment problems by shortest path problems, whenever an assignment is analysed. The result is an alternative to the best complexity algorithm, by Chegireddy and Hamacher, with $\mathcal{O}(Kn^3)$ (considering a network with $2n$ nodes), improving the $\mathcal{O}(Kn^4)$ time complexity of the straightforward implementation of Murty's algorithm. The computational experiments here presented show that this variant outperforms the original algorithm, and one implementation of the algorithm proposed by Chegireddy and Hamacher.

Section 2 gives a brief review of the assignment problem and refers an algorithm to solve it. Section 3 begins with a presentation of the ranking assignments problem and its algorithms, followed by a subsection where the new variant is developed. Computational complexity order is studied. Section 4 shows results of experimental tests for the three algorithms.

2 The assignment problem

Let $(\mathcal{N}, \mathcal{A})$ be a bipartite network where $\mathcal{N} = \mathcal{N}_1 \cup \mathcal{N}_2$, with $\mathcal{N}_1$ and $\mathcal{N}_2$ disjoint sets with n elements each, and where $\mathcal{A} = \mathcal{N}_1 \times \mathcal{N}_2$. Elements in $\mathcal{N}$ are known as nodes and elements in $\mathcal{A}$ as arcs of the network. In the following, nodes in $\mathcal{N}_1$, as well as nodes in $\mathcal{N}_2$, will be denoted by 1, 2,..., n. A non negative integer cost c_{ij} is associated with each arc $(i, j) \in \mathcal{A}$.

The assignment problem arises when trying to pair, with minimum cost, each node in $\mathcal{N}_1$ with a node in $\mathcal{N}_2$. Its linear programming formulation is:

$$\begin{array}{ll} \min & \sum_{(i,j)\in\mathcal{A}} c_{ij}x_{ij} \\ \text{s.t.} & \sum_{j\in\mathcal{N}_2} x_{ij} = 1\,,\ i \in \mathcal{N}_1 \\ & \sum_{i\in\mathcal{N}_1} x_{ij} = 1\,,\ j \in \mathcal{N}_2 \\ & x_{ij} \geq 0\,, \qquad (i,j) \in \mathcal{A}\,. \end{array}$$

A solution x to this problem is called an assignment and will be represented by $a = \{(1, j_1), \ldots, (n, j_n)\}$ (also called assignment), where, for $i \in \{1, \ldots, n\}$, $x_{ij} = \begin{cases} 1\,, \text{ if } j = j_i \\ 0\,, \text{ if } j \neq j_i \end{cases}$. The cost of x is $c(x) = \sum_{(i,j)\in\mathcal{A}} c_{ij}x_{ij}$ or $c(a) = \sum_{(i,j)\in a} c_{ij}$.

The assignment problem can be seen as a minimum cost flow problem in a bipartite network where the capacity upper bound is $u_{ij} = 1$, $(i, j) \in \mathcal{A}$, the supply is $r_i = 1$, $i \in \mathcal{N}_1$, and the demand is $r_j = -1$, $j \in \mathcal{N}_2$. Thus, one of the algorithms known to solve it is the successive use of a shortest path algorithm. It computes a shortest path from a super-source to a super-sink in a residual network and then updates the flow along that path in 1 unit. This corresponds to associate a node in $\mathcal{N}_1$ with another one in $\mathcal{N}_2$, therefore the best assignment can be obtained computing n shortest paths. The complexity of this algorithm is $\mathcal{O}(nc(n))$, where $c(n)$ denotes the number of operations to find a shortest path, thus $\mathcal{O}(n^3)$ using a label setting algorithm. Further details on this algorithm can be found in [11].

Given the flow x in $(\mathcal{N}, \mathcal{A})$, the residual network can be described as $(\mathcal{N}', \mathcal{A}')$ such that:

1. $\mathcal{N}' = \mathcal{N} \cup \{s, t\}$, with $s, t \notin \mathcal{N}$;
2. $\mathcal{A}' = \{(s, i) : i \in \mathcal{N}_1 \wedge x_{ij} = 0, \forall j \in \mathcal{N}_2\} \cup \{(i, j) \in \mathcal{A} : x_{ij} = 0\} \cup \{(j, t) : j \in \mathcal{N}_2 \wedge x_{ij} = 0, \forall i \in \mathcal{N}_1\} \cup \{(j, i) \in \mathcal{A}^r : x_{ij} = 1\}$, where $\mathcal{A}^r = \{(j, i) : (i, j) \in \mathcal{A}\}$ is the set of reverse arcs.

The cost of each (i, j), c'_{ij}, is maintained if $(i, j) \in \mathcal{A}$, is considered 0 for the new arcs starting in s or incident in t, and $c'_{ji} = -c_{ij}$ if $(j, i) \in \mathcal{A}^r$.

3 The ranking assignments problem

In this section the ranking assignments problem is defined, the algorithms for solving it are reviewed, one due to Murty and another due to Chegireddy and Hamacher (further details about these algorithms can be found in [6, 12]). Finally, a variant of Murty's algorithm is presented.

The ranking assignments problem is a generalization of the assignment problem where the aim is to compute the K least cost assignments, with $K > 1$, by non-decreasing order of cost, that is, $a_1, \dots, a_K$, such that:

- $c(a_i) \leq c(a_{i+1})$, for any $i \in \{1, \dots, K - 1\}$;
- a_i is determined immediately before a_{i+1}, for any $i \in \{1, \dots, K - 1\}$;
- $c(a_K) \leq c(a)$, for any other assignment $a \notin \{a_1, \dots, a_K\}$.

In [12] Murty proposed an algorithm to solve this problem. It demands that a routine for solving the assignment problem is known and uses a set X, characterized by storing candidates to a future k-th best assignment, where $k \in \{1, \dots, K\}$. Initially $X = \{a_1\}$. The least cost element is repeatedly picked up from X (being a certain a_k) and analysed in order to generate new "low cost" assignments, then stored in X. The new candidates are found considering a partition of the set of assignments in the network where a_k was computed and solving an assignment problem for every subset in that partition. The subsets to consider are obtained by forcing the inclusion of some a_k arcs (which can be done by avoiding the use of its tail and head nodes) and forbidding others (for instance, replacing its cost by $+\infty$).

Consider a general $k \geq 1$ and the k-th shortest assignment with the form $a_k = \{(i_1, j_1), \ldots, (i_r, j_r), (t_1, s_1), \ldots, (t_{n-r}, s_{n-r})\}$.
Let us assume it is obtained as the best assignment when $(i_1, j_1), \ldots, (i_r, j_r)$ are forced to be in a_k and $(m_1, p_1), \ldots, (m_\ell, p_\ell)$ can't belong to a_k, that is:

- $i_1, \ldots, i_r$ were deleted from $\mathcal{N}_1$; $j_1, \ldots, j_r$ were deleted from $\mathcal{N}_2$, and
- $(m_1, p_1), \ldots, (m_\ell, p_\ell)$ were deleted from $\mathcal{A}$.

After restoring this network, a_k remaining arcs $(t_1, s_1), \ldots, (t_{n-r-1}, s_{n-r-1})$ are successively deleted, thus obtaining networks where no assignment was computed. The assignment problem is solved in each of those networks and each solution is stored in X. In sum, X stores the solutions of the following problems:

1. shortest assignment a such that $(t_1, s_1) \notin a$,
2. shortest assignment a such that $(t_1, s_1) \in a$ and $(t_2, s_2) \notin a$,

...

$n - r - 1$. shortest assignment a such that $(t_1, s_1), \ldots, (t_{n-r-2}, s_{n-r-2}) \in a$ and $(t_{n-r-1}, s_{n-r-1}) \notin a$.

In the worst-case, Murty's algorithm demands the resolution of $n - 1$ assignment problems for each a_k, $k \in \{1, \ldots, K\}$, therefore, using the algorithm in Sect. 2, it has $\mathcal{O}(Kn^4)$ complexity.

Later Hamacher and Queyranne [9] presented a different method to rank solutions of any combinatorial problem. Their method is a generalization of the algorithms by Gabow [8] for ranking spanning trees and by Carraresi and Sodini [5] for ranking simple paths. It uses a binary search tree procedure (so, a partition different from Murty's), based on the computation of the second best solution of the combinatorial problem. Chegireddy and Hamacher [6] used that algorithm to find the K best perfect matchings, proposing several procedures for computing the second best perfect matching in bipartite networks, that is, the second best assignment.

The algorithm of Chegireddy and Hamacher uses a set X, analogously to Murty's algorithm. The algorithm starts by computing the best and the second best assignments, being the last one stored in X. After a_1 is listed, the least cost element in X is repeatedly selected and removed from that set, and then it is analysed. The elements chosen in X are $a_2, \ldots, a_K$. Let a_k be an assignment defined as above, the least cost assignment obtained from a_j, and let (t_e, s_e) be an arc in a_k but not in a_j. Then, under the same computation conditions, two new problems are solved:

- the second best assignment that contains (t_e, s_e);
- the second best assignment that does not contain (t_e, s_e).

Chegireddy and Hamacher proved that finding the second best assignment is equivalent to find the shortest cycle in a residual network relatively to a_k. One of their suggestions consists in computing the shortest path from i to j, for each $(i, j) \in a_k$ in the new network, and, denoting its cost by $d(i, j)$, to select the path such that $d(i, j) + c_{ij}$ is minimum. This demands solving at most n shortest path problems, which is of $\mathcal{O}(n^3)$ if using a label setting algorithm. Therefore, their algorithm has $\mathcal{O}(Kn^3)$ time complexity.

3.1 Variant of Murty's algorithm for ranking assignments

In this subsection a new variant of Murty's algorithm is presented. It differs from the original because it is based on the use of a specific order for solving the successive assignment problems when analysing each k-th assignment, allowing to improve its complexity order.

Given a general a_k, $k \in \{1, \dots, K\}$, the original Murty's algorithm imposes no specific order for solving the successive assignment problems. In the beginning of this section a straightforward implementation of that algorithm was outlined, where those problems were solved, from problems of dimension $n - r$ to problems of dimension 2. In the following the order of analysis will be reversed. In short, it can be said that analysing $a_k = \{(i_1, j_1), \dots, (i_r, j_r), (t_1, s_1), \dots, (t_{n-r}, s_{n-r})\}$, may consist in storing in X each solution of the following problems:

1. shortest assignment a such that $(t_1, s_1), \dots, (t_{n-r-2}, s_{n-r-2}) \in a$ and $(t_{n-r-1}, s_{n-r-1}) \notin a$,

...

$n - r - 2$. shortest assignment a such that $(t_1, s_1) \in a$ and $(t_2, s_2) \notin a$,
$n - r - 1$. shortest assignment a such that $(t_1, s_1) \notin a$.

It should be noticed that the first problem's solution is simply the least cost assignment in a network with $\mathcal{N}_1 = \{t_{n-r-1}, t_{n-r}\}$, $\mathcal{N}_2 = \{s_{n-r-1}, s_{n-r}\}$, and $\mathcal{A} = \mathcal{N}_1 \times \mathcal{N}_2 - \{(t_{n-r-1}, s_{n-r-1})\}$. Furthermore, in this case the dimension of the successive problems is increasing, that is, given a minimum cost assignment in a network, the next assignment to be determined is the best in a new network that differs from the previous by the reinsertion of one pair of nodes and one a_k arc. Hereafter will be shown how some of the information may be used from a problem to the following one, namely the way how a shortest assignment is computed based on previous information and how nodes are restored in the network. The new variant of Murty's algorithm is summarized in Algorithm 1.

Algorithm 1 – *Variant of Murty's ranking assignments algorithm*

```
    $a \longleftarrow$ Minimum cost assignment in $(\mathcal{N}, \mathcal{A})$; $X \longleftarrow \{a\}$
    $k \longleftarrow 0$
    While (($X \neq \emptyset$) and ($k < K$)) Do
        $k \longleftarrow k + 1$
/*  Minimum cost assignment in $X$, computed as the best such  */
/*  that $(i_1, j_1), \dots, (i_r, j_r) \in a_k$ and $(m_1, p_1), \dots, (m_\ell, p_\ell) \notin a_k$  */
        $a_k \longleftarrow \{(i_1, j_1), \dots, (i_r, j_r), (t_1, s_1), \dots, (t_{n-r}, s_{n-r})\}$
        $X \longleftarrow X - \{a_k\}$
        $a' \longleftarrow \{(i_1, j_1), \dots, (i_r, j_r), (t_1, s_1), \dots, (t_{n-r-2}, s_{n-r-2})\}$
/*  Restore $a_k$ computation conditions  */
        Delete $i_1, \dots, i_r, t_1, \dots, t_{n-r-1}$ from $\mathcal{N}_1$ and $j_1, \dots, j_r, s_1, \dots, s_{n-r-1}$
        from $\mathcal{N}_2$
        Delete $(m_1, p_1), \dots, (m_\ell, p_\ell)$ from $\mathcal{A}$
/*  Initialize $b$ with $x_{st_{n-r}} = x_{t_{n-r}s_{n-r}} = x_{s_{n-r}t} = 1$   */
        $b \longleftarrow \{(t_{n-r}, s_{n-r})\}$
        For ($h \in \{1, \dots, n-r-2\}$) Do
            Restore $t_{n-r-h}$ in $\mathcal{N}_1$ and $s_{n-r-h}$ in $\mathcal{N}_2$
            Restore arcs starting in $t_{n-r-h}$ and ending in $s_{n-r-h}$ in $\mathcal{A}$ exclu-
            ding $(t_{n-r-h}, s_{n-r-h})$
            $b \longleftarrow$ Insert nodes $t_{n-r-h}$ and $s_{n-r-h}$ in assignment $b$
            $X \longleftarrow X \cup \{a' \cup b\}$
            Restore $(t_{n-r-h}, s_{n-r-h})$ in $\mathcal{A}$
            $b \longleftarrow \{(t_{n-r-h}, s_{n-r-h}), \dots, (t_{n-r}, s_{n-r})\}$
            $a' \longleftarrow a' - \{(t_{n-r-h-1}, s_{n-r-h-1})\}$
        EndFor
        Restore $t_1$ in $\mathcal{N}_1$ and $s_1$ in $\mathcal{N}_2$
        Restore arcs starting in $t_1$ and ending in $s_1$ in $\mathcal{A}$ excluding $(t_1, s_1)$
        $b \longleftarrow$ Insert nodes $t_1$ and $s_1$ in assignment $b$
        $X \longleftarrow X \cup \{a' \cup b\}$
        Restore $(t_1, s_1)$ in $\mathcal{A}$
    EndWhile
```

Whenever an arc is reinserted, the least cost assignment b coincides with a_k for all non-deleted nodes, therefore we may simply consider that part of a_k. Assume now that b is a best assignment in a given $(\mathcal{N}_1 \cup \mathcal{N}_2, \mathcal{A})$ and suppose i is added to $\mathcal{N}_1$ and i' is added to $\mathcal{N}_2$. Suppose that some arcs are also added, which link i to nodes in $\mathcal{N}_2$ and nodes in $\mathcal{N}_1$ to i'. Theorem 1 allows to conclude that to recalculate b, that is, to compute the best assignment in the modified network, one may compute one shortest path and update the flow in the network, analogously to what is done in the algorithm outlined in Sect. 2 for the assignment problem. Lemma 1 (proved in [1]) supports the proof of Theorem 1.

Lemma 1. *Let b and b' be two assignments in $(\mathcal{N}_1 \cup \mathcal{N}_2, \mathcal{A})$, such that b' is obtained from b by updating the flow along the arcs of an s-t path p in the residual network of $(\mathcal{N}_1 \cup \mathcal{N}_2, \mathcal{A})$. Then $c(b') = c(b) + c(p)$.*

Theorem 1. *Let x be the flow associated with the minimum cost assignment in $(\mathcal{N}_1 \cup \mathcal{N}_2, \mathcal{A})$. The minimum cost assignment in $(\mathcal{N}'_1 \cup \mathcal{N}'_2, \mathcal{A}')$, where:*

- $\mathcal{N}'_1 = \mathcal{N}_1 \cup \{i\}$, $\mathcal{N}'_2 = \mathcal{N}_2 \cup \{i'\}$ *and*
- $\mathcal{A}' = \mathcal{A} \cup \{$*arcs starting in i or ending in i'*$\}$,

can be obtained by computing the shortest path from s to t, and updating the flow, in the residual network of $(\mathcal{N}'_1 \cup \mathcal{N}'_2, \mathcal{A}')$, with $x_{iv} = x_{ui'} = 0$, for any $(i, v), (u, i') \in \mathcal{A}' - \mathcal{A}$.

Proof. Let b be the minimum cost assignment in $(\mathcal{N}_1 \cup \mathcal{N}_2, \mathcal{A})$ and b' be the assignment obtained after updating the flow along the arcs of the shortest path p' from s to t in the residual network of $(\mathcal{N}'_1 \cup \mathcal{N}'_2, \mathcal{A}')$.

Suppose b' doesn't have minimum cost, that is, there exists a b^* such that $c(b^*) \leq c(a)$, for any assignment a, and $c(b^*) < c(b')$.

Let p^* be the s-t path in the residual network of $(\mathcal{N}_1' \cup \mathcal{N}_2', \mathcal{A}')$ with (s, i) as the first arc, (i', t) as the last one, and intermediate arcs that belong, alternatively, to b and b^*. Now it will be shown that $c(b^*) = c(b) + c(p^*)$. Let b^{**} be the assignment obtained from b using p^*. If $b^{**} \neq b^*$ (otherwise that is proved) p^* only affects a subassignment a of b, that is, $b^* = a \cup d^*$, and $b^{**} = a \cup d^{**}$, where $d^* \neq d^{**}$ and d^{**} is also a subassignment of b. Thus, $c(b^*) = c(a) + c(d^*)$ and $c(b^{**}) = c(a) + c(d^{**})$. Since b is the best assignment in $(\mathcal{N}_1 \cup \mathcal{N}_2, \mathcal{A})$, $c(d^*) \leq c(d^{**})$, and, as b^* is the best assignment in $(\mathcal{N}_1' \cup \mathcal{N}_2', \mathcal{A}')$, then $c(d^{**}) \leq c(d^*)$. So, $c(d^*) = c(d^{**})$ and by Lemma 1, $c(b^*) = c(b^{**}) = c(b) + c(p^*)$, which completes the proof.

Also, from Lemma 1, $c(b') = c(b) + c(p')$. Since p' was assumed to be the shortest path from s to t, $c(p^*) \geq c(p')$ and thus $c(b^*) \geq c(b')$, contradicting the assumption made. □

Based on these results the minimum cost assignment can be recalculated as in Procedure 1.1, when two new nodes are added to a given network.

Procedure 1.1. *Insert nodes i and i' in assignment b*

```
    $p \longleftarrow$ Shortest path from $s$ to $t$ in $(\mathcal{N}_1 \cup \mathcal{N}_2, \mathcal{A})$
    If ($\pi_t < +\infty$) Then Update $x$ in 1 unit of flow along arcs of $p$
```

The main advantage of the proposed variant, over the straightforward implementation of Murty's algorithm, is the replacement of successive assignment problem resolutions by considering an initial assignment and then adding nodes and arcs to the network, that is, solving shortest path problems and updating some arc flows. In sum, the analysis of each a_k consists in reoptimising each solution using previous information, thus performing as many operations as when solving a single assignment problem, instead of $n - 1$, as when applying the straightforward implementation. Theorem 2 proves that, in fact, Algorithm 1 has the same worst-case complexity as Chegireddy and Hamacher's algorithm, instead of $\mathcal{O}(Kn^4)$, the complexity order of the original Murty's algorithm.

Theorem 2. *In a worst-case analysis, Algorithm 1 ranks the K least cost assignments in $\mathcal{O}(Kn^3)$ time complexity.*

Proof. In the worst-case each assignment demands the computation of n shortest paths, therefore the complexity of the proposed variant is $\mathcal{O}(Kn\,c(n))$, with $c(n)$ the number of operations needed to find the shortest path. Despite of the existence of arcs with negative cost, the shortest path problems can be solved by an auction algorithm due to Bertsekas et al. [4], with $\mathcal{O}(n^2)$ operations, thus Algorithm 1 has $\mathcal{O}(Kn^3)$ complexity. □

3.2 Example of the variant of Murty's algorithm

Now Algorithm 1 is exemplified by using the network with cost matrix $C = \begin{bmatrix} 0 & 0 & 2 & 2 \\ 1 & 3 & 0 & 4 \\ 0 & 6 & 1 & 0 \\ 3 & 3 & 2 & 0 \end{bmatrix}$. As it has been described the algorithm begins by solving the assignment problem and $X = \{\{(1,2),(2,3),(3,1),(4,4)\}\}$. The unique element of X, a_1, is then selected and three new problems are solved (as shown in Fig. 1 and Table 1).

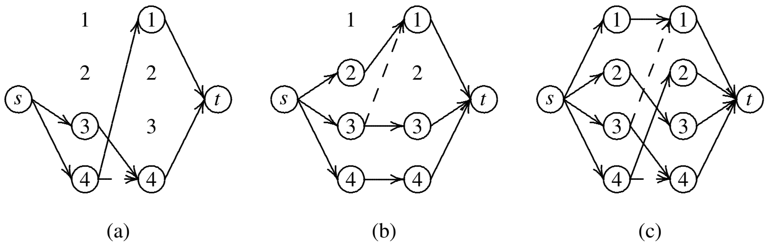

Fig. 1a–c. Minimum cost assignment ranking in $(\mathcal{N}, \mathcal{A})$: (a) Inserting 3 in $\mathcal{N}_1$ and 1 in $\mathcal{N}_2$, with $c_{31} = +\infty$, in assignment $\{(4,4)\}$; (b) Inserting 2 in $\mathcal{N}_1$ and 3 in $\mathcal{N}_2$, with $c_{23} = +\infty$, in $\{(3,1),(4,4)\}$; (c) Inserting 1 in $\mathcal{N}_1$ and 2 in $\mathcal{N}_2$, with $c_{12} = +\infty$, in $\{(2,3),(3,1),(4,4)\}$

Table 1. Ranking assignments – Analysing $a_1 = \{(1,2),(2,3),(3,1),(4,4)\}$.
a_1 computation conditions: No deleted arcs or nodes.
Deleted arcs: $(1,2),(2,3),(3,1),(4,4)$.
Deleted nodes: 1, 2, 3 from $\mathcal{N}_1$, and 1, 2, 3 from $\mathcal{N}_2$

Arcs/nodes added	Computed assignment	Assignment stored in X	c
$(4,4)$ to $\mathcal{A}$ 3 to $\mathcal{N}_1$, 1 to $\mathcal{N}_2$	$\{(3,4),(4,1)\}$	{(1,2),(2,3),(3,4),(4,1)}	3
$(3,1)$ to $\mathcal{A}$ 2 to $\mathcal{N}_1$, 3 to $\mathcal{N}_2$	$\{(2,1),(3,3),(4,4)\}$	{(1,2),(2,1),(3,3),(4,4)}	2
$(2,3)$ to $\mathcal{A}$ 1 to $\mathcal{N}_1$, 2 to $\mathcal{N}_2$	$\{(1,1),(2,3),(3,4),(4,2)\}$	{(1,1),(2,3),(3,4),(4,2)}	3

After that the set of candidates to a next-best assignment is

$$X = \{\{(1,2),(2,3),(3,4),(4,1)\},\{(1,2),(2,1),(3,3),(4,4)\},\\ \{(1,1),(2,3),(3,4),(4,2)\}\}$$

with costs 3, 2 and 3, respectively. In the following step $a_2 = \{(1,2),(2,1),(3,3),(4,4)\}$ is picked from X and an analysis, similar to the previous one, is made, after restoring the conditions in $(\mathcal{N}_1 \cup \mathcal{N}_2, \mathcal{A})$ when a_2 was determined. In this case it consists in deleting $(2,3)$ from $\mathcal{A}$, 1 from $\mathcal{N}_1$, and 2 from $\mathcal{N}_2$. The problems solved when analysing a_2 are shown in Table 2. So,

$$X = \{\{(1,2),(2,3),(3,4),(4,1)\},\{(1,1),(2,3),(3,4),(4,2)\},\\ \{(1,2),(2,1),(3,4),(4,3)\},\{(1,2),(2,4),(3,1),(4,3)\}\}.$$

Table 2. Ranking assignments – Analysing $a_2 = \{(1,2),(2,1),(3,3),(4,4)\}$.
a_2 computation conditions: Arc (2, 3) deleted. Nodes 1 from $\mathcal{N}_1$, and 2 from $\mathcal{N}_2$ deleted.
Deleted arcs: (1, 2), (2, 1), (3, 3), (4, 4).
Deleted nodes: 2, 3 from $\mathcal{N}_1$, and 1, 3 from $\mathcal{N}_2$.

Arcs/nodes added	Computed assignment	Assignment stored in X	c
(4, 4) to $\mathcal{A}$ 3 to $\mathcal{N}_1$, 3 to $\mathcal{N}_2$	$\{(3,4),(4,3)\}$	{(1,2),(2,1),(3,4),(4,3)}	3
(3, 3) to $\mathcal{A}$ 2 to $\mathcal{N}_1$, 1 to $\mathcal{N}_2$	$\{(2,4),(3,1),(4,3)\}$	{(1,2),(2,4),(3,1),(4,3)}	6

4 Computational experiments

In order to evaluate the performance of the new variant of Murty's algorithm from an empirical point of view, several computational experiments comparing it both with its original version and with the one proposed by Chegireddy and Hamacher, have been performed. In this section some of the results obtained in those experiments are reported and analysed.

The algorithms have been implemented in C language originating codes `MA` (for a straightforward implementation of Murty's algorithm), `CHA` (for an implementation of Chegireddy and Hamacher's algorithm), and `VMA` (for an implementation of the new variant of Murty's algorithm). The tests were performed on an AMD Athlon with a 1.5 GHz processor and 512 Mbytes of RAM, running over Linux. The three implementations followed the descriptions in Sect. 3. `MA`, `CHA` and `VMA` were all coded using the same data structure for set X (a maximum storage of K assignments was considered). Moreover, for solving shortest path problems (when finding the best assignment in `MA` and `VMA`, and the 2nd best assignment in `CHA`) the same label correcting algorithm, using a FIFO to store labeled nodes, was applied. A lower worst-case complexity algorithm could have been used in the codes (for instance a label setting one as suggested in [6]). However, label correcting algo-

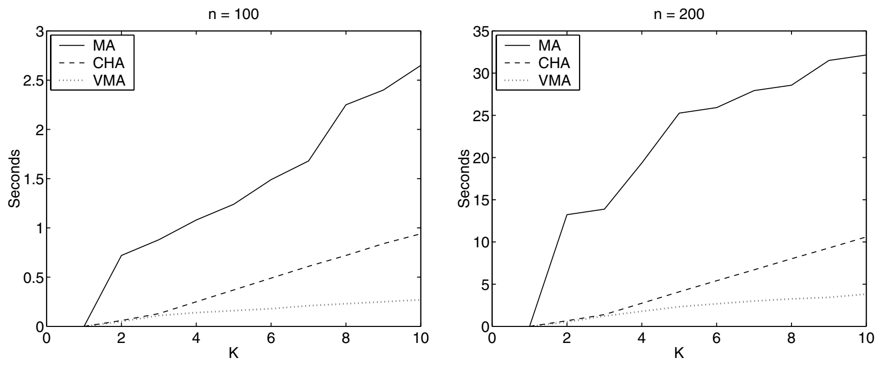

Fig. 2. Test complete bipartite networks, $n = 100$ and $n = 200$

rithms have a good practical performance, moreover, for a proper comparison, all three codes use the same routine to find a shortest path.

A first test-bed consisted of instances `assign100`, ..., `assign600`, drawn from the OR-library available at `http://mscmga.ms.ic.ac.uk/info.html`, by Beasley [2] (see also [3]). Those are complete bipartite network problems with $n \in \{100, \ldots, 600\}$, and for each of these dimensions $K = 10$ assignments were ranked. The correspondent running time results are shown in the plots of Figs. 2, 3 and 4. VMA was the code with best results, followed by CHA and then MA. That code shows an improvement of about 88% relatively to MA CPU times and of 40% to 70% for CHA times. The MA code has a quite big increase after analysing a_1, specially for higher dimension networks, while both CHA and VMA present CPU times linearly dependent on K. One of the reasons why running times grow so quickly, when analysing a_1, and then selecting a_2 in X, may be because this analysis implies solving an increasing number of assignment problems for higher values of n. Besides, at most 10 assignments are stored in X.

Later a more exhaustive study of MA, CHA and VMA's behaviour was made. It used smaller complete bipartite networks with costs uniformly generated in $\{0, \ldots, 1000\}$ and $n \in \{50, 100, 150, 200\}$, but more assignments were ranked ($K = 100$). For each n, 10 problems were generated with different seeds. The average CPU times obtained are shown in the plots of Figs. 5 and 6. The results confirmed the relative behaviours of the three codes.

As expected from the theoretical analysis, MA had a performance worse than the other two codes. As for CHA and VMA, in terms of computational requirements, the overall analysis of each a_k consists in finding two 2nd best assignments in CHA, and one best assignment in VMA. Considering the worst-case, the 2nd best assignment problem demands the computation of $n-1$ shortest paths in a network with $2(n-1)$ nodes. On the other hand, VMA solves n shortest path problems in networks with increasing dimensions, from 4 to $2n$ nodes. This might explain the best empirical

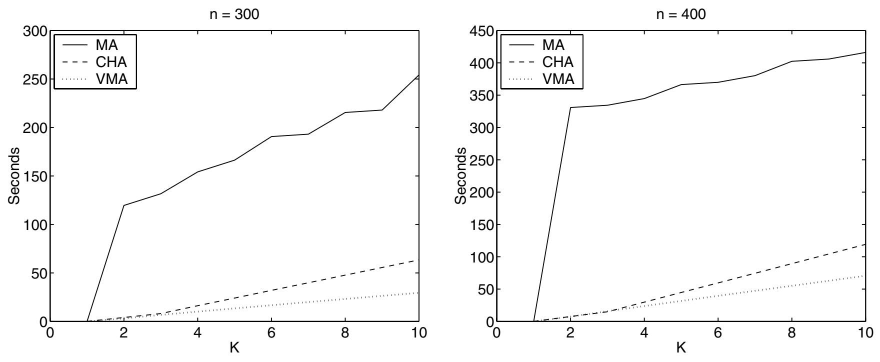

Fig. 3. Test complete bipartite networks, $n = 300$ and $n = 400$

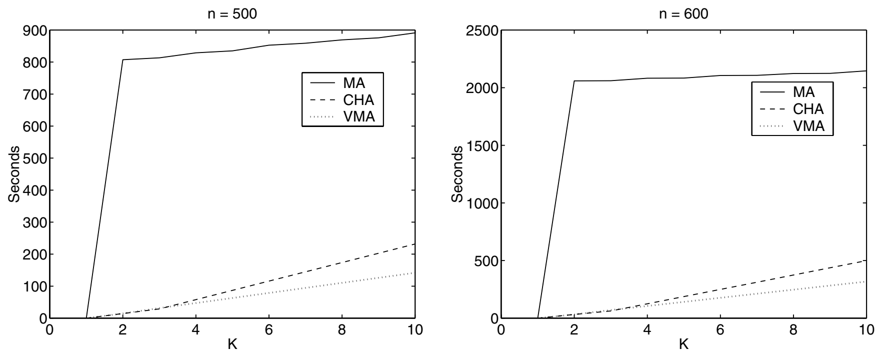

Fig. 4. Test complete bipartite networks, $n = 500$ and $n = 600$

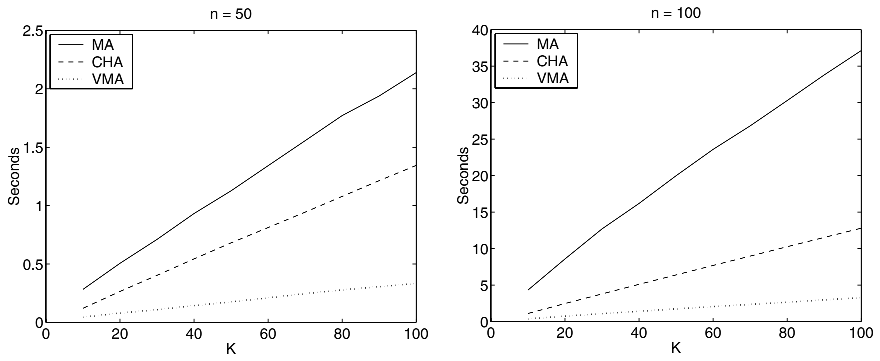

Fig. 5. Complete bipartite networks, $n = 50$ and $n = 100$

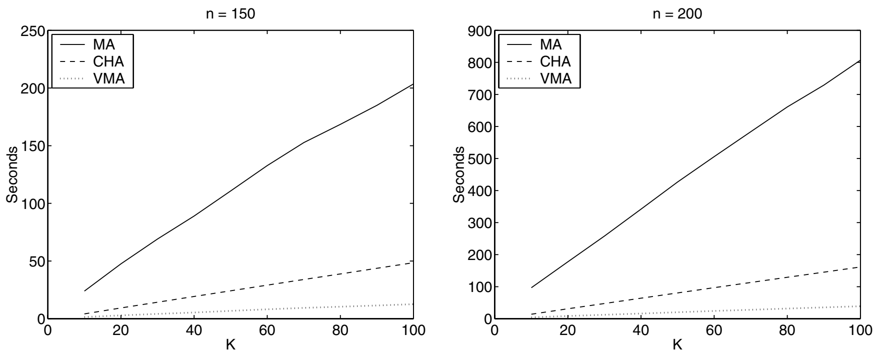

Fig. 6. Complete bipartite networks, $n = 150$ and $n = 200$

performance of VMA relatively to CHA, although their theoretical complexities are the same.

5 Conclusions

Despite of the practical interest in ranking assignments by order of cost, only an algorithm proposed by Murty (with $\mathcal{O}(Kn^4)$) and another one due to Chegireddy and Hamacher (with $\mathcal{O}(Kn^3)$) are known in the literature.

In this work a variant of Murty's algorithm, which leads to a specific implementation of the former algorithm depending on the order used for analysing the arcs of each a_k, $k \in \{1, \ldots, K\}$, was proposed. It was shown that this variant improves the worst case complexity of the former algorithm to $\mathcal{O}(Kn^3)$. Also, comparative computational experiments were presented, which confirm its superiority in practice, both over the original Murty's algorithm and the Chegireddy and Hamacher's algorithm.

References

1. Ahuja RK, Magnanti TL, Orlin JB (1993) *Network flows : Theory, algorithms and applications*. Prentice Hall, Englewood Cliffs, NJ
2. Beasley JE (1990) Linear programming on cray supercomputers. *Journal of the Operational Research Society* 41(2): 133–139
3. Beasley JE (1990) Or-library: distributing test problems by electronic mail. *Journal of the Operational Research Society* 41(2): 1069–1072
4. Bertsekas D, Pallottino S, Scutellà M (1995) Polynomial auction algorithms for shortest paths. *Comp Opt Appl* 4: 99–125
5. Carraresi P, Sodini C (1983) *A binary enumeration tree to find K shortest paths*. In: Proc. 7th Symposium on Operations Research 45 in Methods of Operations Research. Athenäum/Hain/Hanstein, pp 177–188
6. Chegireddy CR, Hamacher HW (1987) Algorithms for finding K-best perfect matchings. *Discrete Applied Mathematics* 18: 155–165

7. Dell'Amico M, Toth P (2000) Algorithms and codes for dense assignment problems: the state of the art. *Discrete Applied Mathematics* 100: 17–48
8. Gabow HN (1977) Two algorithms for generating weighted spanning trees in order. *SIAM Journal on Computing* 6(1): 139–150
9. Hamacher HW, Queyranne M (1985/6) *K-best solutions to combinatorial optimization problems.* (Annals of Operations Research, No. 4) Baltzer Science Publishers, pp 123–143
10. Kuhn HW (1955) The hungarian method for the assignment problem. *Naval Research Logistics Quarterly* 2: 83–97
11. Lawler EL (1976) *Combinatorial optimization: Networks and matroids*. Holt Rinehart and Winston
12. Murty KG (1968) An algorithm for ranking all the assignments in increasing order of cost. *Operations Research* 16: 682–687