Journal of Universal Computer Science, vol. 17, no. 13 (2011), 1854-1862
submitted: 11/3/11, accepted: 29/8/11, appeared: 1/9/11 © *J.UCS*

An Improved FPTAS for Mobile Agent Routing with Time Constraints

Eugene Levner
(Ashkelon Academic College, Ashkelon and Bar Ilan University, Ramat Gan, Israel
elevner@ash-college.ac.il and levnere@mail.biu.ac.il)

Amir Elalouf
(Bar Ilan University, Ramat Gan, Israel
amir.elalouf@biu.ac.il)

T.C.E. Cheng
(The Hong Kong Polytechnic University, Hong Kong
edwin.cheng@inet.polyu.ac.hk)

Abstract: Camponogara and Shima (2010) developed an ε-approximation algorithm (FPTAS) for the mobile agent routing problem in which a benefit function determines how visits to different sites contribute to the agent's mission. The benefit is to be maximized under a time constraint. They reduced the problem to the constrained longest-path problem in a graph. In this note we present a modified FPTAS that improves on their result by a factor of $(n/h)\log\log\left(\bar{b}/\underline{b}\right)$, where $\bar{b}$ and $\underline{b}$ are an upper bound and a lower bound on the maximum benefit, respectively, n is the number of nodes, and h is the length of the longest path (in hops) in the graph.

Keywords: Mobile agent; Constrained routing; Constrained longest path; Approximation algorithm; FPTAS
Categories: F.2.2, G.2.2, I.2.11, J.7

1 Introduction

Camponogara and Shima (2010) developed an efficient ε-approximation algorithm (FPTAS) for the mobile agent routing problem in which computational resources are available at many possible sites. In order to complete its mission, the mobile agent visits a number of hosts that provide information, e.g., sites in the Internet, computer nodes in grids, etc. A given benefit function determines how much benefit (e.g., signal energy, information from sites, retrieval data, etc.) each site contributes to the agent's mission. The problem is to maximize the benefit within a time limit. Since information is available at many different sites that yields different degrees of benefit, the mobile agent should find a best possible itinerary to visit them.

Camponogara and Shima reduce this problem to the constrained longest-path problem (CLPP) in a graph $G = (V, E)$. Their algorithm finds a path with a benefit at least 1-ε times the benefit of the optimal path (such a path is called ε-*approximate*) and has the following property:

The ε-approximation algorithm for CLPP runs in $O(\log\log(\bar{b}/\underline{b})(|E|n/\varepsilon + \log\log(\bar{b}/\underline{b})))$ time, where $\bar{b}$ and $\underline{b}$ are an upper bound and a lower bound on the maximum benefit, respectively, and n and $|E|$ are the number of nodes and arcs in the underlying directed graph $G = (V, E)$, respectively.

One of the main difficulties in obtaining an FPTAS for the considered problem is a lack of simply computable tight lower and upper bounds. Camponogara-Shima's algorithm (CSA), in which the term $\underline{b}$ depends on arc benefits, can be classified as a "weakly polynomial" FPTAS. The main theoretical contribution of this note is that we derive a "strongly polynomial" FPTAS that runs in $O(|E|h/\varepsilon + |E|h\log\log h)$ time, thus reducing the complexity of CSA by a factor of $(n/h)\log\log(\bar{b}/\underline{b})$, where h denotes the length of the longest path (in hops) in graph G.

2 Outline of Camponogara-Shima's Algorithm

We begin with a brief description of the algorithm by Camponogara and Shima (2010). We use the same notation as introduced by them. Arcs in G are denoted by (i,j), b_{ij} is the benefit of traversing arc (i,j), r_{ij} is the time resource consumption to move along arc (i,j), r is the amount of resource available, and $|V| = n$. Benefit $b(p)$ and time resource $r(p)$ of a route p from a given node s to a given node t are defined as the sum of the benefits and times, respectively, of the arcs in the path. A directed path from node i to node j (denoted as the i-j path) is called a b-path if its benefit is at least b and an r-path if its time consumption is not greater than r.

Like many other FPASs available in the literature, CSA comprises three basic sub-procedures as the building blocks:

(a) Program Test(v) that answers the question "$b^* \prec v$?", where b^* is the maximum benefit value within the time limit, and v is a given parameter. The test outputs "yes" if $b^* \prec v$ and outputs "no" if $b^* \succeq v(1-\varepsilon)$. It is a dynamic-programming-based procedure of complexity $O(|E|n/\varepsilon)$, where $n = |V|$.

(b) Scaling of arc benefit values.

(c) A dynamic programming (DP) procedure, called dual-Reverse-DP(b) that computes a maximum-benefit r-path from s to t in $O(|E|\bar{b})$.

We do not include the details of the DP algorithm as it is a standard procedure well known in the literature. However, in order to make the paper self-contained, we present the original procedure Test(v) and the ε–approximation algorithm on the scaled data called FPTAS($\bar{b}$, $\underline{b}$, φ, ε, r) in Figures 1 and 2, respectively. They will be modified and their novel combination will lead to our new algorithm with the improved time complexity.

Test (v)
1. Take a fixed ε, $0 < \varepsilon < 1$. Scale (i.e., round up) the benefit values b_{ij} and replace them by $\lceil \min(b_{ij}, v)/(v\varepsilon/(n-1)) \rceil$.
2. Run **Dual-Reverse-DP** over graph G with scaled benefits as follows:
Compute the time resource consumption $r_i(b)$ of a b-path from node i to sink t with the least consumption, which is defined recursively by:

$$r_i(b) = \min_{(i,j)\in E} \left\{ r_j \left(\max\{b - b_{ij}, 0\} \right) + r_{ij} \right\}, b = 0, 1, \ldots, K$$

until $r_s(b) > r$ for some $b < (n-1)/\varepsilon$, in which case every b-path has a benefit at most $b^* < v$,
or else $b \geq (n-1)/\varepsilon$, in which case an r-path with a benefit of at least $v(1-\varepsilon)$ is found.
Here b increases from 0 up to some K until one of the above conditions is satisfied.

Figure 1: Test procedure by Camponogara and Shima (2010)

FPTAS($\bar{b}$, $\underline{b}$, φ, ε, r)
1: **while** $\bar{b}/\underline{b} > \varphi$, do
2: $v = \sqrt{\bar{b} \cdot \underline{b}}$
3: **if** Test(v) ="yes" then
4: $\bar{b} \leftarrow v$
5: **else**
6: $\underline{b} \leftarrow v(1-\varepsilon)$
7: **end if**
8: **end while**
9: Set $b_{ij} \leftarrow \lceil b_{ij}/(\underline{b}\varepsilon/(n-1)) \rceil$
10: Apply **Dual-Reverse-DP** to obtain an optimal r-path

Figure 2: Camponogara and Shima's (2010) ε–approximation procedure

3 The Improved FPTAS

In the improved ε-approximation algorithm, we modify the scaling and testing sub-procedures of CSA and exploit them in a different way. Our approach basically follows a computational scheme first suggested by Gens and Levner (1981) for the min-cost knapsack problem. It consists of three main stages:

Stage A. Find a preliminary lower bound LB and preliminary upper bound UB on the optimal solution such that $UB/LB \leq h \leq n-1$.

Stage B. Find improved lower and upper bounds $\bar{\beta}$ and $\underline{\beta}$ on the optimal solution such that $\bar{\beta}/\underline{\beta} \leq 2$.

Stage C. Scale the input data and find an ε-approximation solution using a dynamic programming sub-algorithm on the scaled data.

Stages B and C are modifications of the two stages in CSA depicted in Figures 1 and 2, while Stage A is totally absent from CSA. These changes permit us to essentially improve the complexity of the algorithm. As we will see below, in practice h can be essentially smaller than n. The computation of h is clarified below in Step A3 in Section 3.1.

The logic of the modified algorithm is the following.

First, the preliminary bounds LB and UB (such that $UB/LB < h$) are found in Stage A. After that, the algorithm runs the binary search in a logarithmic scale in a similar way as it is performed by Hassin (1992) and Camponogara and Shima (2010) (see Figure 1). After each test run, the bounds are modified. Each current test is executed at the point $v = (UB*LB)^{1/2}$. However, there is a difference between our modified test, denoted by M-Test(v), and the original Test(v). This difference helps accelerate the algorithm. We will explain this in detail below.

When Stage B terminates, we obtain improved lower and upper bounds, and then use them in the modified ε-approximation procedure in Stage C. The main difference between the new ε-approximation procedure in Stage 3 and **FPTAS**($\bar{b}$, $\underline{b}$, φ, ε, r) is that factor h is used instead of n-1 in the scaling. Specifically, instead of the scaling operation indicated in Step 9 in Figure 2, we use the following one:

$$\text{Set } b_{ij} \leftarrow \left\lceil \frac{b_{ij}}{\underline{\beta} \cdot \varepsilon / h} \right\rceil \tag{1}$$

where h denotes the length of the longest (in hops) s-t path in graph G.

Proposition 1. Scaling in stage C according to (1) increases each arc benefit by at most $\underline{\beta} \cdot \varepsilon / h$ and each path benefit by at most $\underline{\beta} \cdot \varepsilon \le b^* \varepsilon$.

The proof can be conducted along the same line as the corresponding proofs for similar statements in Gens and Levner (1981), Hassin (1992), and Camponogara and Shima (2010). In fact, we just substitute n-1 by h.

Notice that replacing (n-1) by h in (1) does not improve the worst-case complexity but dramatically accelerates computation in practical computer networks.

The number of iterations in Stage B needed to decrease the ratio UB/LB from h to a value below 2 is evidently (log log h). We will show that, in contrast to Hassin's and Camponogara-Shima's procedures, each of which needs to run Test(v) O(log log(UB/LB)) times (where each test requires, in turn, $O(|E|h/\varepsilon)$ time), our modification scheme needs only O(log log h) runs of the modified test M-Test(v), wherein each test requires $O(|E|h)$ time. The time to compute the test point $b = (UB*LB)^{1/2}$ before each test is O(log log(UB/LB)) in Camponogara-Shima's Test(v) and O(log log h) in the modified test M-Test(v), which we will describe below. We have arrived at our key observation:

Proposition 2.The modified test M-Test(v) for CLPP requires O(log log h) iterations, each of which requires $O(|E|h)$ time to run the test operations and O(log log h) time more to compute the test point. Thus, M-Test(v) runs in O(log log h ($|E|h$ + log log h)) = $O(|E|h$ log log h) time.

Taking into account that Stage A requires $O(|E|+n)$ time and Stage C requires $O(|E|h/\varepsilon)$ time, we obtain the following improved complexity for the modified algorithm:

Proposition 3. The modified algorithm for CLPP requires a total of $O(|E|h/\varepsilon + |E|h\log\log h)$ time, thus improving the complexity of CSA by a factor of $(n/h)\log\log(\bar{b}/\underline{b})$.

We present the proofs in the following.

3.1 Description of Stage A

Notice that graph G is acyclic since we are solving the longest path problem (with respect to benefit), with non-negative benefit values serving as arc "lengths". At the same time, in order to find the preliminary lower and upper bounds, we will solve the shortest (with respect to time) path problem, in which the arc times serve as the arc "lengths". After obtaining the shortest (with respect to time) paths from the soruce to each node and from each node to the destination (Step A1 below), we can determine the minimum-time path passing through any arc (Step A2). Then we will select an appropriate arc with the maximum benefit value (Step A3). More specifically, we carry out the following four steps:

Step A1. Find the finite shortest (with respect to time) *s-i* paths and shortest (with respect to time) *j-t* paths for all those arcs (i, j) for which such paths exist. For this aim, we run twice the standard algorithm for finding the shortest path from one source to all the destinations in the acyclic directed graph G; during these runs, we treat node s as a source in the first run and node t as a "source" in the second run. In the second run we change the arc direction.

Step A2. For each arc (i,j) in G, add the length of the shortest (in time) *s-i* path, the length of the shortest (in time) *j-t* path, and the length $r(i, j)$ of arc (i, j) itself. Denote the obtained sum by $R(i, j)$.

Step A3. Let us call the arc (i,j) *right* if $R(i, j) \le r$. (W.l.o.g., we assume that at least one such an arc does exist in G). Scan all the "right" arcs and select among them one with the maximum benefit value. Denote the latter value by b_0. Clearly, $b_0 \le b^* \le nb_0$. Since any *s-t* path contains at most h hops, the following also holds: $b_0 \le b^* \le hb_0$. Therefore, we have found $LB = b_0$ and $UB = hb_0$.

Step A4. Find h, the number of hops in the longest (with respect to hops) *s-t* path in graph G. For this aim, we run the standard algorithm for finding the longest *s-t* path in the acyclic directed graph G.

Complexity of Stage A. The complexity of Steps A1 and A4 is $O(|E|+n)$ (see Cormen et al., 2001, ch.24.2) and the complexity of Steps A2 and A3 is $O(|E|)$. Therefore, Stage 1 requires $O(|E|+n)$ time in total.

3.2 Description of Stage B

The new test denoted by M-Test(v) is based on Test(v) described in Figure 1. The difference between them is two-fold. First, instead of scaling the benefits as indicated in Step 1 in Figure 1, we use a different scaling operation defined as follows:

$$b_{ij} \leftarrow \left\lceil \frac{b_{ij}}{v\varepsilon/h} \right\rceil .$$

Second, when computing the resource consumption $r_i(b)$ of a b-path from node i to sink t, we again substitute n-1 by h. Specifically, $r_i(b)$ is computed recursively, like in Step 2 in Figure 1, for $b = 0, 1, \ldots$, until $r_s(b) > r$ for some $b < h/\varepsilon$, in which case every b-path has a benefit at most $b^* < v$, or else $b \geq h/\varepsilon$, in which case an r-path of benefit at least $v(1-\varepsilon)$ is found.

Thus, similar to Test(v), **M-Test**(v) answers "yes" if $b^* < v$ or "no" if $b^* \geq v(1-\varepsilon)$. One can easily verify that this testing property still holds after changing n-1 to h. Due to this modification, the complexity of the testing procedure decreases from $O(|E|n/\varepsilon)$ to $O(|E|h/\varepsilon)$ time. In addition, in the sub-algorithm **Bounds** described below, we use the modified **M-Test**(v) only for several fixed ε-values (ε = 1/4, 1/3, and 1/2). Due to this, the complexity of **M-Test**($v|_{\varepsilon \text{ fixed}}$) decreases from $O(|E|h/\varepsilon)$ to $O(|E|h)$ time; on the other hand, the latter algorithm guarantees the same testing property as Test(v).

The sub-algorithm **BOUNDS** provides the improved lower and upper bounds $\underline{\beta}$ and $\overline{\beta}$ such that $\overline{\beta}/\underline{\beta} \leq 2$. It is presented in Figure 3. In this figure, **M-Test**($v|_{\varepsilon=0.5}$) denotes M-Test(v) carried out with fixed ε = 0.5. Obviosly, **M-Test**($v|_{\varepsilon=0.5}$) either returns $b^* < v$ or, otherwise, returns $b^* \geq 0.5v$. Similarly, **M-Test**($v|_{\varepsilon=1/3}$) either reports $b^* < v$ or, else, $b^* \geq 2/3v$. **M-Test**($v|_{\varepsilon=0.25}$) either reports $b^* < v$, or, else, $b^* \geq 0.75v$.

Bounds
Input: *LB* and *UB* such that *UB/LB* ≤ n*h*.
Output: $\overline{\beta}$ and $\underline{\beta}$ such that $\overline{\beta}/\underline{\beta} \leq 2$

1. If *UB/LB* ≤ 2 goto 6
2. Set $v = 2\sqrt{LB \cdot UB}$
3. Run Test1 = **M-Test** ($v|_{\varepsilon = 0.5}$)
   if Test1="*no*" set *LB*← 0.5v and goto 1
   if Test1="*yes*" set *UB* ←v
4. Set $v \leftarrow v/2$; Run Test2 = **M-Test**($v|_{\varepsilon=1/3}$)
   if Test2 = "*yes*" set *UB*← v and goto 1
   if Test2= "*no*" set *LB* ←2/3v (now the ratio *UB/LB* = 3)
5. Set v← 2*LB*. Run Test3=**M-Test**($v|_{\varepsilon=0.25}$)
   if Test3="*yes*" set *UB*←v
   if Test3= "*no" set LB*← 0.75v
   (now the ratio *UB/LB* = 2)
6. $\underline{\beta}$ = *LB*, $\overline{\beta}$ = *UB* end

*Figure 3: Sub-algorithm **Bounds***

The proof of the validity of **Bounds** is straightforward and is omitted here.

Complexity of Stage B. Since (i) the test point in any step of **Bounds** is $v = \sqrt{LB \cdot UB}$, (ii) **M-Test**($v|_{\varepsilon \text{ fixed}}$) runs in $O(|E|h)$ time, and (iii) the number of returns to Step 1 in **Bounds** is at most log log h, the overall complexity of Stage B is $O(|E|$h log log $h)$.

Notice that complexity of Stage C is discussed in the beginning of Section 3 and Proposition 3. The logical structure of the two algorithms is presented in Figure 4.

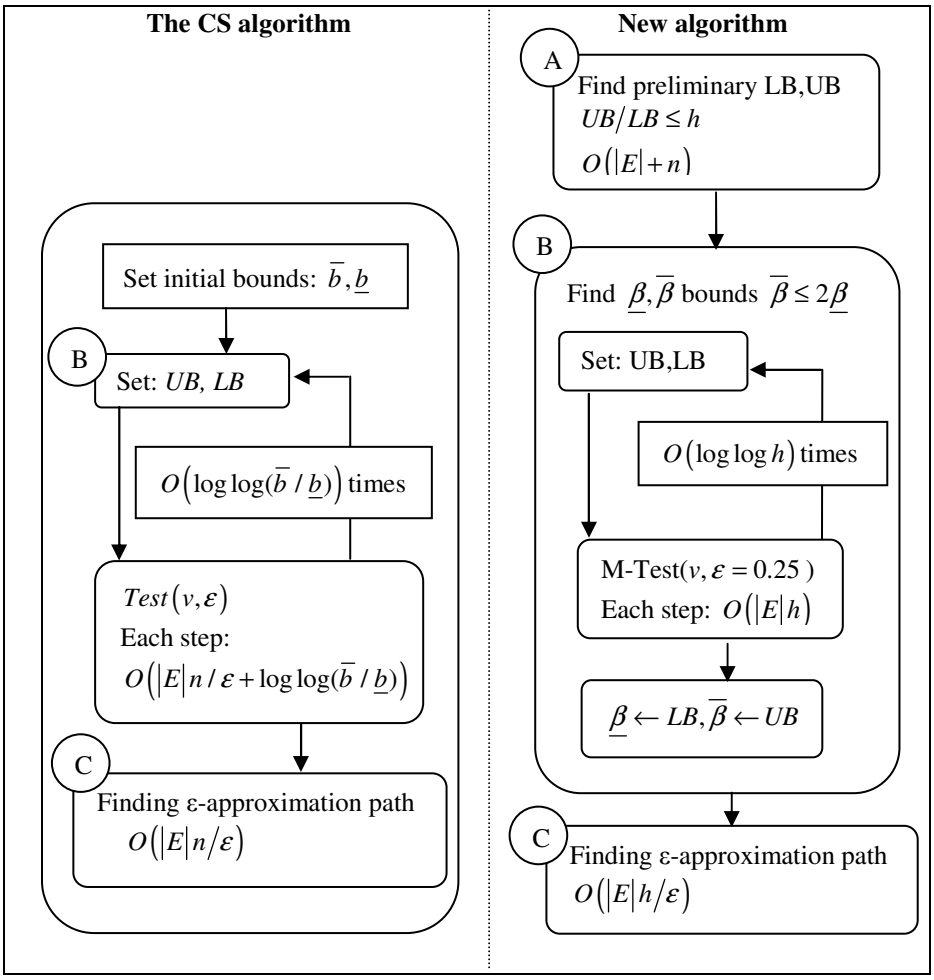

Figure 4: The structure of two algorithms

4 Summary and Conclusions

According to Camponogara and Shima (2010), the motivation for agent mobility and efficient agent routing is not in the technology itself but rather in the benefit that agents offer for innumerous applications in communications, computing systems, and the Internet, as well as in educational, social, and cultural services. The mobile agent itinerary problem can be cast as the resource-constrained longest-path problem that appeals for fast and practical algorithms.

Table 1 summarizes the main differences between CSA and our new algorithm.

Algorithm Stages	**CSA**	**New algorithm**
Stage A	None	Use special algorithm for initial *UB, LB* to achieve a ratio equal to *h*.
Complexity of Stage A	–	$O\left(\lvert E\rvert + n\right)$
Stage B	Call DP with a given ε	Call DP with a constant ε, equal to 1/2, 1/3, and ¼.
The number of iterations	$O\left(\log\log(\bar{b}\,/\,\underline{b})\right)$	$O\left(\log\log(h)\right)$ as a result of the improvement in Stage A.
Complexity at each iteration	$O\left(\lvert E\rvert n/\varepsilon + \log\log(\bar{b}\,/\,\underline{b})\right)$	$O\left(\lvert E\rvert h\right)$ as a result of a constant ε
Total complexity of Stage B	$O\left(\begin{array}{l}\left(\log\log(\bar{b}\,/\,\underline{b})\right)\bullet \\ \left(\lvert E\rvert n/\varepsilon + \log\log(\bar{b}\,/\,\underline{b})\right)\end{array}\right)$	$O\left(\log\log h \bullet \left(\lvert E\rvert h\right)\right)$
Stage C	Scaled DP with paramater *n*	Scaled DP with parameter *h*
Complexity of Stage C	$O\left(\lvert E\rvert n/\varepsilon\right)$	$O\left(\lvert E\rvert h/\varepsilon\right)$
Total complexity of all stages A,B and C	$O\left(\begin{array}{l}\left(\log\log(\bar{b}\,/\,\underline{b})\right)\bullet \\ \left(\lvert E\rvert n/\varepsilon + \log\log(\bar{b}\,/\,\underline{b})\right)\end{array}\right)$	$O\left(\log\log h \bullet \lvert E\rvert h + \lvert E\rvert h/\varepsilon\right)$

Table 1: Comparison of the algorithms

In this note we suggest a modified algorithm that improves the original Componogara-Shima algorithm by a factor of (*n/h*)log log($\bar{b}$ /$\underline{b}$). Therefore, the complexity of our algorithm polynomially depends only on the graph size (i.e., |*E*| and *n*) and 1/ε, and does not depend on the magnitudes of the input data. In addition, although the maximum number of hops, *h*, in graph *G*, in the worst case can be *n*-1, the factor *n/h* in realistic computing networks may be up to dozens and even hundreds. Our simulation experiments with agents serving computer communication

networks confirm that the running time of the new technique is considerably less than that of the CSA. Figure 5, borrowed from Lennon and Maurer (1994), illustrates this fact. In this example, the average node out-degree in the network is just 4 (though, in practice, it may be much larger), so in the 5-layer network the acceleration factor *n/h* is larger than 600.

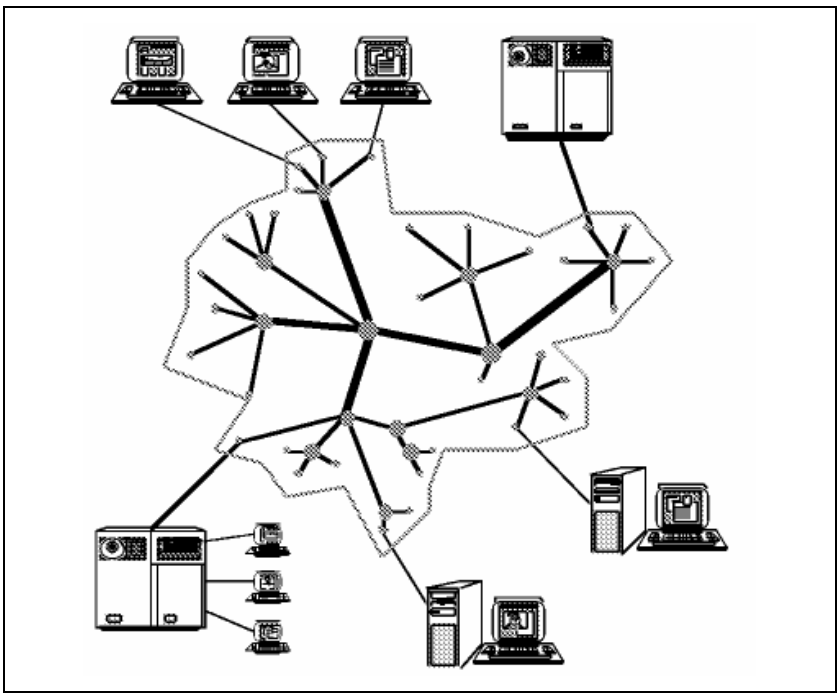

Figure 5: A networked multimedia system (Lennon and Maurer, 1994)

References

[Camponogara and Shima, 2010] Camponogara, E., Shima, R. B. (2010). Mobile agent souting with time constraints: a resource constrained longest-path approach. *Journal of Universal Computer Science*, 16, 3 372-401.

[Cormen et al. 2001] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. (2001). *Introduction to Algorithms*, MIT Press, Cambridge, USA.

[Gens and Levner 1981] Gens, G. V., Levner, E. V. (1981). Fast approximation algorithms for job sequencing with deadlines. *Discrete Applied Mathematics*, 3 313-318.

[Hassin 1992] Hassin, R. (1992). Approximation schemes for the restricted shortest path problem. *Mathematics of Operations Research*, 17, 1 36-42.

[Lennon and Maurer 1994] Lennon, J., Maurer, H. (1994). Applications and impact of hypermedia systems: an overview. *Journal of Universal Computer Science*, pilot issue, 54-108.