Journal of Universal Computer Science, vol. 16, no. 3 (2010), 372-401
submitted: 9/10/08, accepted: 29/12/09, appeared: 1/2/10 © *J.UCS*

Mobile Agent Routing with Time Constraints: A Resource Constrained Longest-Path Approach

Eduardo Camponogara and **Ricardo Boveto Shima**
(Department of Automation and Systems Engineering
Federal University of Santa Catarina
88040-900 Florianópolis, SC, Brazil
camponog@das.ufsc.br, ricardo.shima@gmail.com)

Abstract: Mobile agent technology advocates the mobility of code rather than the transfer of data. As data is found in several sites, a mobile agent has to plan an itinerary to visit several sites where it collects resources to accomplish its mission. This gives rise to the mobile-agent itinerary problem (MIP) which seeks a route maximizing overall benefit from the resources while meeting a deadline. This paper formalizes MIP and develops a reduction to the resource constrained longest-path problem (CLPP) in acyclic graphs. A dynamic programming (DP) algorithm was designed to produce a family of optimal routes, allowing a mobile agent to dynamically revise its route. A fully-polynomial approximation scheme was developed to reduce the pseudo-polynomial running time of DP, whereby the distance to the optimal is controlled by a parameter ϵ and the running time is limited by a polynomial on problem size and $1/\epsilon$. The paper reports results from experiments assessing the performance of the algorithms and discusses extensions to handle non-additive objectives, non-additive constraints, and probabilistic resource constraints.
Key Words: mobile agents, constrained routing, constrained longest-path, dynamic programming, approximation algorithms
Category: F.2.2, G.2.2, I.2.11

1 Introduction

The Internet has grown to become the world's largest repository of information and communication media for millions of people [Kotz and Gray 1999]. Countless companies rely on the Internet to conduct business, including financial markets, the entertainment industry, and the services industry to name a few. Today, information is dynamic, distributed over the network, and its access varies in time and space. To this end, the mobile agent technology advocates the mobility of code rather than data to cope with large volumes of data and discontinuous network connections.

A mobile agent is not bound to the system on which it begins its execution, being free to reach other hosts in the network [Lange and Oshima 1998]. A mobile agent is defined as a self-contained piece of software responsible for the execution of a task. It can transport its code (the static part) and state (the dynamic and time-varying part) to another environment. The state contains all of the necessary information and variable values that enable the agent to resume its execution. According to [Lange and Oshima 1999], the motivation for agent mobility is not the technology itself but rather the benefits agents offer for creating and operating distributed systems. They list seven reasons for using mobile agents, in particular: they overcome network latency, reduce network load, and are naturally heterogeneous.

In typical applications, a mobile agent visits several hosts in a network to complete a task, the so-called agencies [Erfurth and Rossak 2003]. Agencies are dedicated platforms that provide information, from local sensors and data repositories, and also offer services such as specialized algorithms, computational facilities, and environments for inter-agent communication. As services and information necessary to accomplish a task are available at several sites, in different forms, levels of accuracy, and reliabilities, a mobile agent has to plan an itinerary to visit the agencies. Mobile agents should be able to compute and revise itineraries in response to the dynamic behavior of modern networks, and change of information and state of agencies. Itinerary computation is not a single, but rather an evolving family of problems that are somewhat dependent on the target application.

[Brewington et al. 1999] present a distributed system for information retrieval in which a mobile agent visits a sequence of machines until the requested information is found. Each machine takes a given time to process the agent's task and has a probability of being successful, namely a probability of retrieving the desired information. There are travel times from one machine to another. Itinerary computation consists in finding a route that minimizes the expected travel time to complete the task. As itinerary computation underlies the traveling salesman problem, these authors propose simplifications that allow the design of polynomial-time dynamic programming algorithms.

[Erfurth and Rossak 2003] propose a framework for itinerary computation under uncertainty. Given a general task, a mobile agent consults service maps and assembles a list of sites to visit in order to complete its task. The agent produces an itinerary by solving shortest-path and traveling-salesman problems that may visit remote sites. Because the maps on the remote sites can be outdated and imprecise, the agent solves the problem approximately with heuristics and revises the itinerary as it travels through the network.

[Wu et al. 2004] address the routing of mobile agents to fuse data in distributed sensor networks. Rather than concentrating sensor signals at a single point, a mobile agent traverses the network incrementally fusing data until a level of accuracy is reached. The authors consider the energy consumption at the sensor nodes, the path loss associated with data transmission between nodes, and the signal energy which is proportional to the energy emitted by the target. The authors develop an analytic expression for each of these three criteria as a function of the nodes and the order in which they are visited. The routing problem is cast as a combinatorial path problem that, in a single objective, maximizes signal energy while minimizing path loss and energy consumption. A genetic algorithm is proposed to circumvent the computational hardness of the problem.

[Rech et al. 2005, Rech et al. 2006] focus on the computation of an itinerary to enable an agent to achieve its mission while respecting a time deadline. The mission is characterized by a subset of resources that the agent should collect from the network servers while respecting a partial order. Resource is an abstract concept that could mean processing capacity, a data base, and a device, among other things. The network servers provide resources of varying kinds and qualities, giving rise to the problem of deciding which servers and the order in which they should be visited to maximize mission quality, but without violating the deadline. These authors state an itinerary computation problem and propose some heuristics. Applications of their framework are found in the electric-power grid, in which a mobile agent performs fault diagnosis by collecting data from geographically distributed sites and running a decision-making engine [Tolbert et al. 2001].

This paper is related to the mobile agent framework of [Rech et al. 2005]. Specifically, it contributes to mobile agent technology by developing a graph-theoretic model for itinerary computation under time constraints that enables the design of provably optimal, dynamic-programming (DP) algorithms and approximation algorithms. Section 2 formally states the mobile-agent itinerary problem (MIP) and gives a reformulation as a (resource) constrained longest-path problem (CLPP). Section 3 briefly discusses algorithms for CLPP and presents a dual, reverse tree, dynamic-programming algorithm which allows a mobile agent to dynamically adapt to stochastic variations on travel and processing time. Section 4 reports results from computational experiments designed to assess the performance of the DP and approximation algorithms. Section 5 outlines a number of extensions from this work, including the treatment of non-additive objective functions, non-additive constraints, and probabilistic constraints. Section 6 gives a summary of the paper.

2 Mobile Agent Routing with Time Constraints

2.1 Problem Set-Up

The mobile agent framework from [Rech et al. 2005] is based on a computational system consisting of a set N of nodes and a set R of resources. Each node $i \in N$ provides a subset $R_i \subseteq R$ of resources. The illustrative scenario given in Fig. 1 has $N = \{n_0, n_1, \ldots, n_6\}$ and $R = \{r_0, r_1, \ldots, r_8\}$. Node n_0 and resource r_0 are dummy objects not appearing in the figure which will be discussed later. Notice that resource r_1 appears in more than one node, $r_1 \in R_{n_1}$ and $r_1 \in R_{n_3}$, whereas resource r_8 is available only at node 5, $r_8 \in R_{n_5}$.

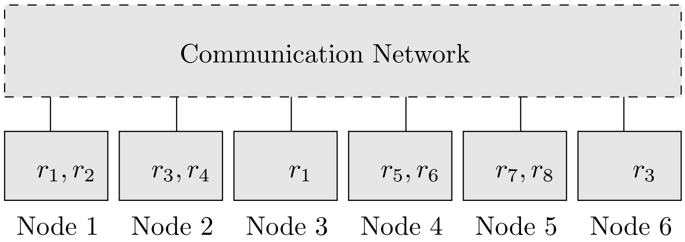

Figure 1: Resources provided by the computational system of the sample scenario. Source: [Rech et al. 2005].

Each mobile agent has a *mission* M defining the resources and the sequence in which they should be collected to accomplish its task. A benefit function $b : R \times N \times \mathbb{R}_+ \to \mathbb{R}_+$ determines how each resource $r \in R$ contributes to the mission depending on the node and time usage. The *resource diagram* of a mission M is modeled by a UML activity diagram defining the resources and the order in which they must be used (precedence relations). Resources other than those appearing in the resource diagram or otherwise disrespecting the order add no value to the mission.

The agent starts its mission at a dummy node labeled n_0 where only a dummy resource r_0 is avaliable. Sometimes these dummy objects will be omitted from diagrams and formulas to simplify notation. The resource diagram of mission M in the illustrative scenario appears in Fig. 2. Notice that the agent uses the dummy resource r_0 first. Then it uses resources r_1 and r_2 in this order. The stereotype $\langle\langle$variable$\rangle\rangle$ means that the benefit from using resource r_1 varies according to time usage. The agent has to use resources r_3 and r_4, in this order, and resource r_5. The diagram does not show any precedence between r_5 and the other two resources, so r_5 may be used before r_3, in between r_3 and r_4, or after r_4. However, these three resources must be used before the agent uses resource r_6. After r_6, the agent is free to use or discard the optional resources r_7 and r_8. The final resource is r_0 which forces the agent to return to its origin.

Given N, R, and the resource diagram, the *node diagram* of a mission M is obtained by replacing each resource in the resource diagram by the nodes where it is found in the computational system. The resulting diagram is a UML activity diagram. Fig. 3 depicts the node diagram for mission M of the sample scenario.

As resources are found in different nodes with potentially different benefits, the problem rests on computing an itinerary for the mobile agent to collect resources according to the resource diagram that maximizes the cumulative benefit, while meeting a time deadline d. An *itinerary* I defines the sequence of nodes visited by the mobile agent and the resources used in each one of the

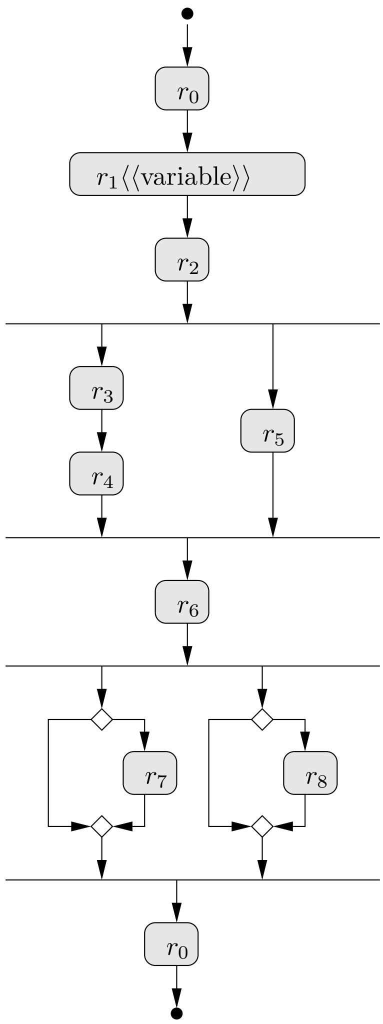

Figure 2: Resource diagram of mission M. Source: [Rech et al. 2005].

them. Formally, an itinerary $I = \langle(i_1, k_1, t_1), (i_2, k_2, t_2), \ldots\rangle$ gives a sequence of nodes, resources to be used, and time spent at each node. For the illustrative example, a feasible itinerary is $I = \langle(n_1, r_1, t_{n_1 r_1})$, $(n_1, r_2, t_{n_1 r_2})$, $(n_2, r_3, t_{n_2 r_3})$, $(n_2, r_4, t_{n_2 r_4})$, $(n_4, r_5, t_{n_4 r_5})$, $(n_4, r_6, t_{n_4 r_6})$, $(n_5, r_8, t_{n_5 r_8})\rangle$ which traces a path in the resource diagram (Fig. 2) that is compatible with the node diagram (Fig. 3). t_{ir} is the processing time for maximum benefit of resource r at node i. An itinerary does not necessarily traverse all nodes, but some nodes may be visited more than once. Notice that the itinerary does not include the dummy node n_0

at which the agent's mission starts and finishes.

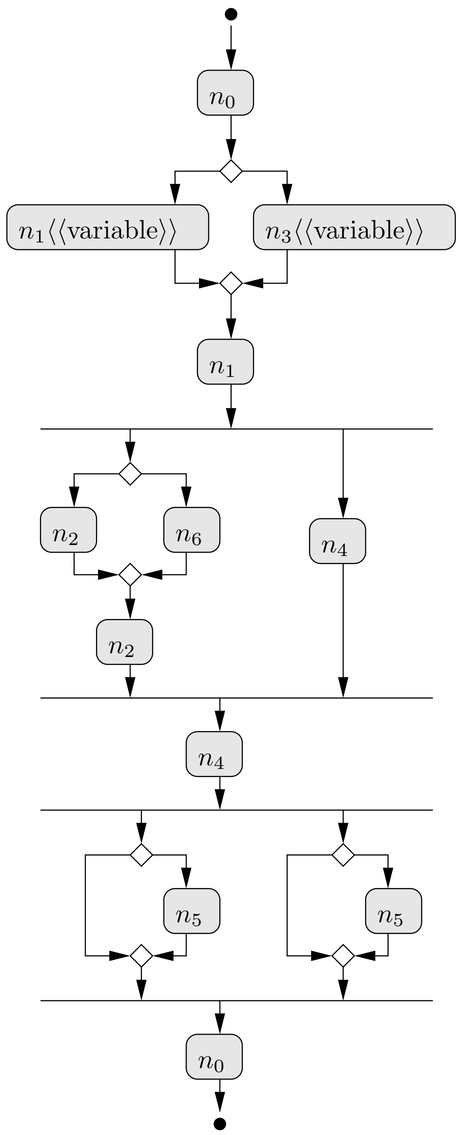

Figure 3: Node diagram of mission M. Source: [Rech et al. 2005].

The execution of a mission M corresponds to an aperiodic activity that occurs sporadically. The processing time of a mission is a direct function of its itinerary, accounting for the network latency l_{ij} for the agent to migrate from a node i to j and the time t_{ir} for maximum benefit of resource r at node i. A *normal resource* r adds the total benefit b_{ir} if the agent stays at i for a time equal to or greater than t_{ir}. On the other hand, a *variable resource* r adds a benefit proportional to the visit time t given by $b_{ir}\min(t/t_{ir}, 1)$. The deadline d determines the time by which the agent should finish its mission. In the illustrative scenario, the latencies are all unitary including node n_0 and the benefits are given in Tab. 1. As the illustrative itinerary consumes $l_{n_0n_1} + l_{n_1n_2} + l_{n_2n_4} + l_{n_4n_5} + l_{n_5n_0} = 5$ time units in latency and $t_{n_1r_1} + t_{n_1r_2} + t_{n_2r_3} + t_{n_2r_4} + t_{n_4r_5} + t_{n_4r_6} + t_{n_5r_8} = 10+2+7+2+8+5+3 = 37$ to use resources, the total time to finalize the mission is $t(I) = 42$ which induces a total benefit $b(I) = 10+3+3+7+5+3+8 = 39$.

Table 1: Time required and benefit induced by resources.

Resource (r_k)	Node (n_i)	Benefit ($b_{n_ir_k}$)	Time ($t_{n_ir_k}$)	Resource (r_k)	Node (n_i)	Benefit ($b_{n_ir_k}$)	Time ($t_{n_ir_k}$)
r_1	n_1	10	10	r_4	n_2	7	2
r_1	n_3	10	10	r_5	n_4	5	8
r_2	n_1	3	2	r_6	n_4	3	5
r_3	n_2	3	7	r_7	n_5	6	7
r_3	n_6	3	7	r_8	n_5	8	3

Having introduced the notation, the mobile-agent itinerary problem (MIP) consists in finding a feasible itinerary that maximizes the cumulative benefit while meeting the time deadline. By a feasible itinerary we mean a node-resource-usage sequence I that uses resources as defined by the resource diagram and visits the nodes where they are found. The expressions:

$$b(I) = \sum_{i_j \in I} \{b_{i_jk_j} \min(1, \lfloor t_j/t_{i_jk_j} \rfloor) : k_j \text{ is not variable}\} + \sum_{i_j \in I} \{b_{i_jk_j} \min(1, t_j/t_{i_jk_j}) : k_j \text{ is variable}\} \tag{1a}$$

$$t(I) = l_{n_0i_1} + \sum_{j=1}^{|I|-1} l_{i_ji_{j+1}} + l_{i_{|I|}n_0} + \sum_{j=1}^{|I|} t_j \tag{1b}$$

give the total benefit and the mission time. Formally, the problem is cast as:

$$\begin{aligned} MIP: \quad & \text{Maximize} \quad b(I) \\ & \text{S.to:} \quad t(I) \le d \\ & \qquad\quad\; I \text{ is feasible} \end{aligned}$$

The notation and problem formulation presented heretofore are contributions of this paper. It formalizes the somewhat imprecise problem statement and solution definition put forward in [Rech et al. 2005, Rech 2006]. For instance, the itinerary was given only by the node sequence which is not sufficient to fully represent a solution.

MIP is an NP-Hard problem since the standard knapsack problem is reducible to MIP in polynomial time and size [Garey and Johnson 1979, Shima 2006]. Consider a knapsack problem of capacity b with n items, each with a weight w_j and value a_j. It suffices to associate each item j with a node n_j and a resource r_j: $R = \{r_j : j = 1, \ldots, n\}$, $N = \{n_j : j = 1, \ldots, n\}$, and $R_j = \{r_j\}$. The benefit of resource r_j is $b_{n_j r_j} = a_j$ and its processing time at n_j is $t_{n_j r_j} = a_j$. By defining the resource diagram as a sequence optionally visiting each node, the latencies $l_{ij} = 0$, and the deadline $d = b$, a solution to MIP yields the solution to the knapsack problem.

2.2 Heuristics for the Mobile-Agent Itinerary Problem

Owing to the computational hardness of the mobile-agent itinerary problem, MIP, a few heuristics appeared in the literature to produce approximate solutions [Rech et al. 2005, Rech et al. 2006, Rech 2006]. Brief descriptions of these heuristics follow below.

Lazy Heuristic: it seeks for the quickest route by ignoring resource benefits. Resources with stereotype ⟨⟨variable⟩⟩ are executed as quickly as possible. The branch of smallest execution time is chosen whenever there are alternatives. The mobile agent transfers itself to the closest node only if the current node does not support the resource needed at the moment.

Greedy Heuristic: its behavior is the opposite of the lazy heuristic, choosing always the alternative that offers the greatest benefit regardless of the execution time. Resources labeled with stereotype ⟨⟨variable⟩⟩ are executed to maximum benefit and when a resource is not available at the current node the mobile agent migrates to nearest node that offers the resource.

Random Heuristic: when a resource required according to the resource diagram is not available at the current node, the agent chooses the next node at random. Resources with variable benefit are utilized for a random time.

Highest Density Heuristic: when there is a choice of the next benefit to be collected, the mobile agent chooses the pair benefit-node with the highest ratio benefit to execution time, the so-called density. Suppose the current node is n_i. The density of a resource r_k available at a node n_j is defined by $db_{n_j r_k} = b_{n_j r_k}/(t_{n_j r_k} + l_{n_i n_j})$ where $b_{n_j r_k}$ is the benefit induced by resource

r_k when processed at node n_j for a time $t_{n_j r_k}$. To decide upon utilizing optional resources, the mobile agent keeps track of the mean value *db* of densities observed thus far and collects an optional resource r_k at a node n_j only if $db_{n_j r_k} \geq db$.

Clock Versions: the heuristics above can be augmented to decide upon the execution of optional resources based on the time remaining until the deadline. Optional resources are collected only if the remaining time is greater than the time required to collect the mandatory resources, including the time necessary to move from one node to another. Except for the lazy heuristic, clock versions can be devised for all of the other heuristics.

The descriptions above serve as guidelines for implementing heuristics. There are several ways of designing heuristics, combining them with local improvement procedures, and even resorting to meta-heuristics [Glover and Kochenberger 2003]. Regardless of the arrangement of such heuristics, they can fail to find a feasible let alone an optimal itinerary even in simple instances of MIP. The baseline heuristics (without clock) yielded the itineraries given in Tab. 2 for the sample instance. The baseline heuristics are insensitive to a time deadline.

Table 2: Itineraries computed by baseline heuristics.

Heuristic	Itinerary (I)	$t(I)$	$b(I)$
Lazy	$\langle(n_1, r_1, 0), (n_1, r_2, t_{n_1 r_2}), (n_6, r_3, t_{n_6 r_3}), (n_2, r_4, t_{n_2 r_4}),$ $(n_4, r_5, t_{n_4 r_5}), (n_4, r_6, t_{n_4 r_6})\rangle$	29	21
Greedy	$\langle(n_1, r_1, t_{n_1 r_1}), (n_1, r_2, t_{n_1 r_2}), (n_4, r_5, t_{n_4 r_5}), (n_6, r_3, t_{n_6 r_3}),$ $(n_2, r_4, t_{n_2 r_4}), (n_4, r_6, t_{n_4 r_6}), (n_5, r_7, t_{n_5 r_7}), (n_5, r_8, t_{n_5 r_8})\rangle$	51	45
Random	$\langle(n_3, r_1, 10), (n_1, r_2, t_{n_1, r_2}), (n_4, r_5, t_{n_4 r_5}), (n_2, r_3, t_{n_2 r_3}),$ $(n_2, r_4, t_{n_2, r_4}), (n_4, r_6, t_{n_4 r_6}), (n_5, r_7, t_{n_5 r_7})\rangle$	48	37
Density	$\langle(n_1, r_1, t_{n_1 r_1}), (n_1, r_2, t_{n_1 r_2}), (n_4, r_5, t_{n_4 r_5}), (n_2, r_3, t_{n_2 r_3}),$ $(n_2, r_4, t_{n_2 r_4}), (n_4, r_6, t_{n_4 r_6}), (n_5, r_8, t_{n_5 r_8})\rangle$	43	39

2.3 Routing as a (Resource) Constrained Longest-Path Problem

Here, we show that the mobile-agent itinerary problem is reducible to the (resource) constrained longest-path problem by generating a suitable, acyclic *mission graph.* This enables us to develop effective dynamic-programming (DP) algorithms that yield provably optimal itineraries for a range of time deadlines (constraints) and DP-based approximation algorithms that trade-off computational time and solution quality. Further, these algorithms can handle benefits and execution times modeled as random variables, as well as additive and non-additive (min-max) constraints and objectives. The reduction is presented in a somewhat informal manner by illustrating its principles in the sample instance.

Fig. 4 depicts the *mission graph* $G = (V, E)$ for the sample instance given by the resource diagram of Fig. 2, the node diagram of Fig. 3, and the parameters from Tab. 1. The nodes are not uniquely labeled to keep the presentation intuitive. The principles of the reduction rest on representing each node n_i of the node diagram by two nodes, n_i and n'_i, and an arc (n_i, n'_i) to which is associated a benefit $b_{n_i r_j}$ for using resource r_j at this node during a time $t_{n_i r_j}$. So, the agent is effectively using resource r_j when traversing arc (n_i, n'_i). Since the mission graph is acyclic, MIP can be cast as a resource constrained shortest-path problem by taking the arc costs as the negative of their benefits [Ziegelmann 2001, Xiao et al. 2005, Shima 2006]. Good previous work on the constrained shortest-path problem is done by [Joksch 1966], [Handler and Zang 1980], [Beasley and Christofides 1989], [Mehlhorn and Ziegelmann 2000]. Let us explain the structure of the mission graph for the sample instance:

1. Starting from node n_0, the mobile agent may move to node n_1 or node n_3 to collect the variable resource r_1. Either way, the agent collects no benefit in the migration but spends 1 unit of time as modeled by the pair $(0, 1)$ associated with the arcs (n_0, n_1) and (n_0, n_3).

2. At node n_1 (equivalently at n_3), the agent can use resource r_1 from 0 to 10 time units and accumulate a proportional benefit as it moves from node n_1 to n'_1. The graph presents only three of these options, namely the arc $(0, 0)$ with the decision of not using resource r_1, the arc $(5, 5)$ in which case the agent uses r_1 for 5 time units, and the arc $(10, 10)$ with maximum benefit.

3. Resource r_2 is only available at node n_1, so if the agent is already at node n_1 it spends no time transferring to n_1 as represented by the arcs with label $(0, 0)$ from n'_1 to n_1. On the other hand, if the agent is at node n_3, it spends one time unit and receives no benefit to move from n_3 to n_1, which is modeled by the arcs (n'_3, n_1) with label $(0, 1)$. At node n_1, the agent has no choice but to collect the resource r_2 which takes 2 units of time and accrues 3 units of benefit. This is modeled by arc (n_1, n'_1).

4. At node n_1, the mobile agent should collect resources r_3, r_4, and r_5. Resource r_3 should be collected before r_4, but r_5 can be obtained before r_3, in between r_3 and r_4, or after r_4. The agent can move to node n_2 or n_6 to collect resource r_3, or move to node n_4 to collect resource r_5. Any of these moves accrues no benefit but takes 1 unit of time.

   Suppose the agent moves to node n_4 to collect resource r_5, which accrues a

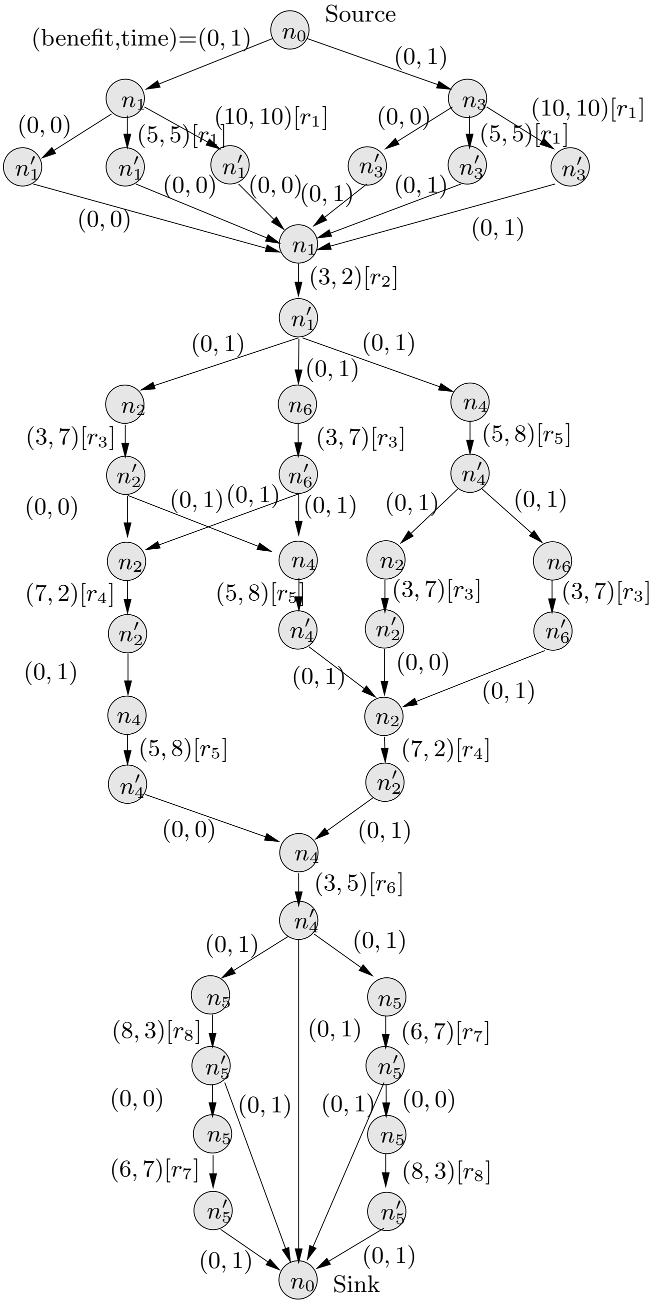

Figure 4: Mission graph for the sample instance.

benefit of 5 and consumes 8 units of time as modeled by arc (n_4, n_4'). The agent then has to collect resource r_3 by moving to either node n_2 or n_6, two options that appear in the graph. After collecting resource r_3, which accrues 3 units of benefit and takes 7 units of time in either node, the agent remains at or moves to node n_2 to collect resource r_4.

The other two options, where the agent collects resource r_5 in between r_3 and r_4 or after r_4, are also covered in the mission graph.

5. In the same manner, one can verify that the mission graph represents the different ways of collecting the remaining resources: r_6, r_7 (optional), and r_8 (optional). The terminal node of the graph is n_0.

Clearly, an itinerary for the mobile agent to accomplish the mission is one-to-one related to a path from the top node n_0 *(source)* to the bottom node n_0 *(sink)*. Given a deadline d, the problem consists in finding a longest-path from the source to the sink that does not consume more time than d. Or, equivalently, a shortest-path in the same graph but with negative benefits. A path from source to sink uniquely defines a sequence of nodes, resources, and time usages that make up an itinerary for the mobile agent. For instance, the leftmost path in Fig. 4 yields the itinerary $I = \langle(n_1, r_1, 0)$, $(n_1, r_2, 2)$, $(n_2, r_3, 7)$, $(n_2, r_4, 2)$, $(n_4, r_5, 8)$, $(n_4, r_6, 5)$, $(n_5, r_8, 3)$, $(n_5, r_7, 7)\rangle$, where $t(I) = 39$ and $b(I) = 35$.

A relative simple algorithm can be implemented to generate the mission-graph from the resource diagram, node diagram, and problem parameters. Such algorithm would combine a breadth-first search with a combination-generation procedure, which would handle the processing of resources that are not related by precedence constraints, such as r_5 and the set $\{r_3, r_4\}$ in Fig. 2. The computational complexity of the mission-graph generation algorithm is a direct function of the number of resources that can be processed in any order. Thus, mission-graph generation and itinerary computation via constrained longest-path are only effective if the number of parallel activities are relatively small.

3 Algorithms for the Constrained Longest-Path Problem

The focus here is on the modeling and design of algorithms for the constrained longest-path problem as a means to solve the mobile-agent itinerary problem. In the presence of only one resource constraint, the (resource) constrained longest-path problem is cast in mathematical programming as:

$$P: \text{ Max} \quad \sum_{(i,j)\in E} b_{ij}x_{ij} \tag{2a}$$

$$\text{S.to}: \quad \sum_{(i,j)\in E} r_{ij}x_{ij} \leq r \tag{2b}$$

$$\sum_{j:(i,j)\in E} x_{ij} - \sum_{j:(j,i)\in E} x_{ji} = \begin{cases} 1, & \text{if } i = s \\ 0, & \text{if } i \in V - \{s,t\} \\ -1, & \text{if } i = t \end{cases} \tag{2c}$$

$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in E \tag{2d}$$

where: $G = (V, E)$ is a directed graph; $b_{ij} \in \mathbb{Z}_+$ is the benefit of traversing arc (i, j); x_{ij} is a decision variable that takes on value 1 if (i, j) is part of the longest path, otherwise it takes on value 0; r_{ij} is the resource consumption to move along arc (i, j) where $r_{ij} \in \mathbb{Z}_+$; r is the amount of resource available; s is the source node; and t is the sink or destination node. An illustrative instance of CLPP appears in Fig. 5, where each arc (i, j) is labeled with the benefit and resource-consumption pair (b_{ij}, r_{ij}) and the available resource is $r = 11$. For applications to the mobile-agent itinerary problem and the development of algorithms, G is assumed to be acyclic. Further, $V = \{1, \ldots, n\}$, the source is $s = 1$, the sink is $t = n$, and the vertices are numbered according to a topological order of G, meaning that $i < j$ if there exists a path from i to j.

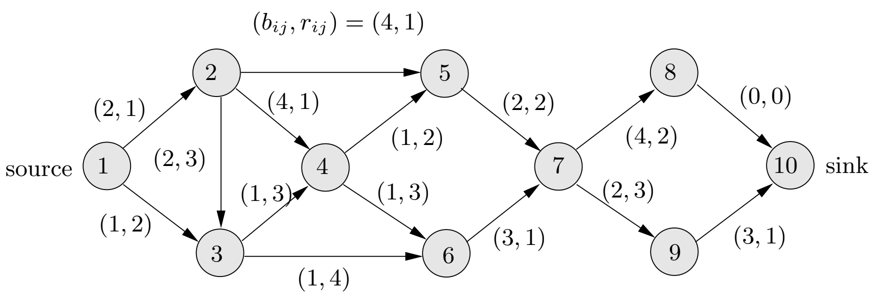

Figure 5: Illustrative instance of the constrained longest-path problem, where the resource available is $r = 11$.

3.1 Dynamic Programming Algorithm

Dynamic programming is a powerful technique for solving sequential decision-making problems, often encountered in control engineering and computer science

[Bertsekas 1995, Cormen et al. 2001]. It breaks up the problem into a sequence of related sub-problems that are solved recursively with less effort than tackling the whole problem. A key property of dynamic programming is that it produces globally optimal solutions when suitably applied. Another property is that DP yields the optimal solution for a family of problems, namely problems where the available resource varies from none to the maximum. For the problem at hand, there are two choices regarding the application of the DP framework:

Longest-path tree: the longest paths can be from source s to the other nodes or from the nodes to sink t. The first strategy is called *direct tree* whereas the second, *reverse tree*. For applications to mobile-agent technology, the reverse tree is preferred as it allows the agent to dynamically adapt to variation in resource usage as it moves along its itinerary.

Objective: assuming the reverse tree structure, the primal approach computes the maximum benefit $b_i(r)$ to reach sink t from a node i given that r units of resource are available. The dual approach computes the minimum resource $r_i(b)$ necessary to reach sink t from a node i with a benefit of at least b. Both approaches lead to pseudo-polynomial time algorithms, with the running time of the primal dependent on the available resource r, whereas the dual is dependent on the optimal objective or an upper bound. Notice that rounding and scaling of benefits compromise only the benefit not resource feasibility in dual form, making the dual formulation suitable for approximation algorithms.

This paper presents only the dual, reverse tree formulation as it allows dynamic adaptation and the design of approximate algorithms. The other formulations are akin to those for the resource constrained shortest-path problem found in [Shima 2006]. A path from node i to sink t is said to be a b-path if its benefit is at least b and an r-path if its resource consumption is not in excess of r. Let $r_i(b)$ be the resource consumption of a b-path, from node i to sink t, of least consumption which is defined recursively by:

$$r_i(b) = \min_{(i,j)\in E}\{r_j(\max\{b - b_{ij}, 0\}) + r_{ij}\} \tag{3}$$

Let $b^\star$ be the value of an optimal solution to problem P. The goal is to find the $b^\star$-path of least resource consumption from source $s = 1$ to sink $t = n$. By setting:

$$r_i(b) = \begin{cases} 0, & \text{if } i = t \text{ and } b = 0 \\ \infty, & \text{if } i = t \text{ and } b > 0 \\ \infty, & \text{if } i \neq t \text{ and } b > b^\star \end{cases} \tag{4}$$

and recursively applying recursion (3), the dual of the constrained longest-path problem is solved with the optimal solution given by $r_1(b^\star)$. Although $b^\star$ is not known a priori, the DP algorithm accepts an upper bound $\bar{b}$ on $b^\star$ for which halting conditions are devised below. The DP algorithm for the dual, reverse tree form is:

Dual-Reverse-DP$(\bar{b})$

for $b = 1$ to $\bar{b}$ **do**
    $r_n(b) \leftarrow \infty$
    $\pi_n(b) \leftarrow$ NIL
end for
$r_n(0) \leftarrow 0$
$\pi_n(0) \leftarrow$ NIL
for $b = 0$ to $\bar{b}$ **do**
    for $i = n - 1$ downto 1 **do**
        $r_i(b) \leftarrow \infty$
        $\pi_i(b) \leftarrow$ NIL
        for $j : (i, j) \in E$ **do**
            if $r_{ij} + r_j(\max\{b - b_{ij}, 0\}) < r_i(b)$ **then**
                $r_i(b) \leftarrow r_{ij} + r_j(\max\{b - b_{ij}, 0\})$
                $\pi_i(b) \leftarrow j$
            end if
        end for
    end for
end for

$\pi_i(b)$ gives the successor of node i in the minimum resource consumption b-path from node i to sink t. The running time of Dual-Reverse-DP is clearly $\Theta(|E|\bar{b})$ and its memory usage is $\Theta(n\bar{b})$ to store tables $r_i(b)$ and $\pi_i(b)$.

3.1.1 Example

Dual-Reverse-DP was applied to the instance of Fig. 5, which yielded Tab. 3 with the minimum resource consumptions and Tab. 4 with the successor pointers. The length of the longest, unconstrained path (with respect to resource) from node 1 to the sink defined the upper bound $\bar{b} = 15$. According to Tab. 3, $b = 15$ is the least benefit for which $r_1(b) > r = 11$, implying that $b^\star = b - 1 = 14$. The optimal path from node 1 to sink t is $\langle 1, 2, 4, 6, 7, 8, 10 \rangle$ which consumes $r_1(14) = 10$ units of resource and induces a benefit of $b^\star = 14$ units.

Fig. 6 depicts the reverse, longest-path tree. It gives the longest-path of minimum resource consumption from each node to the sink for varying benefits. For node 6, path $\langle 6, 7, 8, 10 \rangle$ induces a benefit of 7 units when the resource availability is 3 or 4 units. There does not exist a path from node 6 to 10 with a

Table 3: Dynamic programming table $r_i(b)$ for sample instance.

$r_i(b)\backslash b$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$r_1(b)$	8	8	8	8	8	8	8	8	8	8	8	8	8	10	10	12
$r_2(b)$	5	5	5	5	5	5	5	5	5	5	5	7	7	9	∞	∞
$r_3(b)$	7	7	7	7	7	7	7	7	7	9	11	∞	∞	∞	∞	∞
$r_4(b)$	6	6	6	6	6	6	6	6	6	8	∞	∞	∞	∞	∞	∞
$r_5(b)$	4	4	4	4	4	4	4	6	∞	∞	∞	∞	∞	∞	∞	∞
$r_6(b)$	3	3	3	3	3	3	3	3	5	∞	∞	∞	∞	∞	∞	∞
$r_7(b)$	2	2	2	2	2	4	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
$r_8(b)$	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
$r_9(b)$	1	1	1	1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
$r_{10}(b)$	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Table 4: Dynamic programming table $\pi_i(b)$ for sample instance.

$\pi_i(b)\backslash b$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi_1(b)$	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
$\pi_2(b)$	5	5	5	5	5	5	5	5	5	5	5	4	4	4	-	-
$\pi_3(b)$	6	6	6	6	6	6	6	6	6	4	4	-	-	-	-	-
$\pi_4(b)$	5	5	5	5	5	5	5	5	6	6	-	-	-	-	-	-
$\pi_5(b)$	7	7	7	7	7	7	7	7	-	-	-	-	-	-	-	-
$\pi_6(b)$	7	7	7	7	7	7	7	7	7	-	-	-	-	-	-	-
$\pi_7(b)$	8	8	8	8	8	9	-	-	-	-	-	-	-	-	-	-
$\pi_8(b)$	10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
$\pi_9(b)$	10	10	10	10	-	-	-	-	-	-	-	-	-	-	-	-
$\pi_{10}(b)$	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

resource consumption smaller than 3 units. With a resource availability of 5 or more units, path $\langle 6, 7, 9, 10 \rangle$ gives the maximum benefit of 8 units.

3.1.2 Halting Conditions ($b^\star$)

The maximum benefit of a resource-feasible path is $b^\star = \max\{b : r_1(b) \leq r\}$. The halting condition for the dynamic-programming algorithm is satisfied by the least b for which $r_1(b) > r$, meaning that the optimal path from source to sink induces a benefit $b^\star = b - 1$ and consumes $r_1(b-1)$ units of resource. Thus, Dual-Reverse-DP can be easily modified to run until this halting condition is met, thereby making the algorithm run in time $\Theta(|E|b^\star)$ and using $\Theta(nb^\star)$ units of memory.

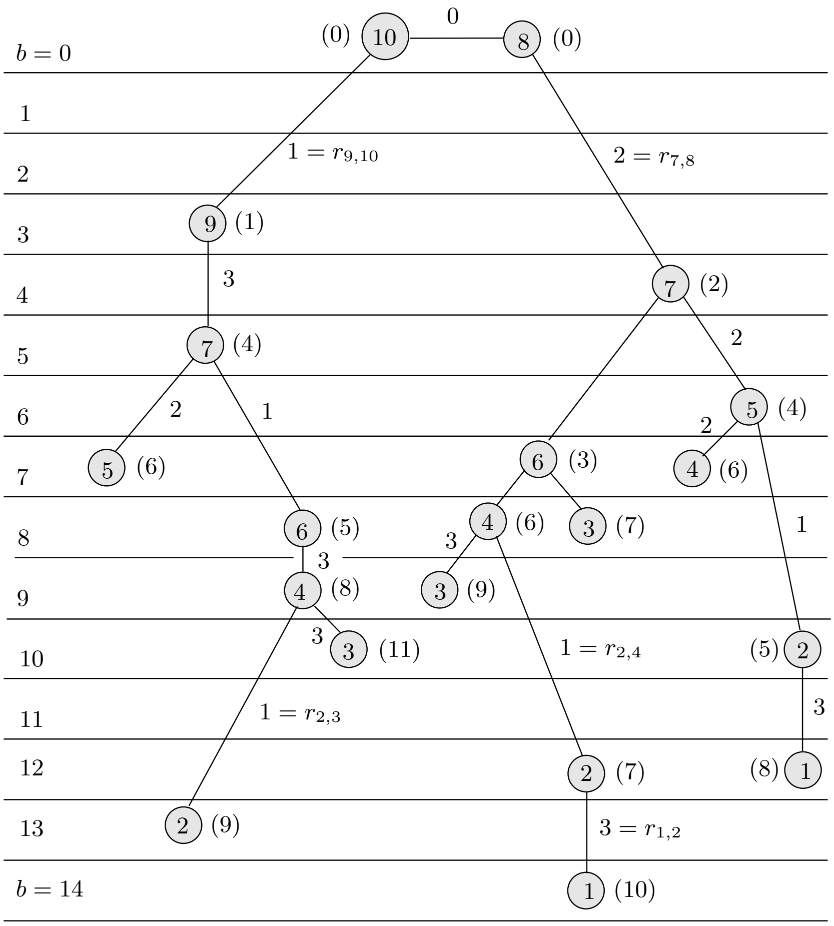

Figure 6: Reverse longest-path tree for the illustrative instance.

A straightforward bound for $b^\star$ is $\bar{b} = (n-1)\max\{b_{ij} : (i,j) \in E\}$, as the longest path from source to sink cannot use more than $(n-1)$ arcs, each of which with a benefit not greater than the maximum arc benefit. A tighter bound is the length of the longest, unconstrained path from source to sink. It is computed in $\Theta(|E|)$ time using a shortest-path algorithm for acyclic graphs with negative benefits as the arc costs [Cormen et al. 2001].

3.2 Approximation Algorithm

An algorithm that returns a solution whose distance to the optimum is controlled by a parameter ϵ is called an *approximation algorithm* and it is said to be a *fully* *polynomial time approximation scheme* (FPTAS) if its running time is bounded by a polynomial on the size of the problem instance and $1/\epsilon$. An FPTAS for the resource constrained longest-path problem is proposed below based on the algorithm design principles from [Sahni 1977, Hassin 1992].

Take a fixed ϵ, $0 < \epsilon < 1$. A polynomial-time ϵ-approximation test is develop below to answer the question "$b^\star < v$?" given some v. The test outputs *"yes"* if $b^\star < v$. It outputs *"no"* if $b^\star \geq v(1-\epsilon)$. The test rounds up the benefit values b_{ij}, replacing them by:

$$\left\lceil \frac{b_{ij}}{v\epsilon/(n-1)} \right\rceil$$

Notice that the original objective of P is related to the objective with scaled benefit values as follows:

$$\begin{aligned} \max \sum_{(i,j)\in E} b_{ij}x_{ij} &\leq \max \sum_{(i,j)\in E} \left\lceil \frac{b_{ij}}{v\epsilon/(n-1)} \right\rceil \frac{v\epsilon}{(n-1)} x_{ij} \\ &= \frac{v\epsilon}{(n-1)} \max \sum_{(i,j)\in E} \left\lceil \frac{b_{ij}}{v\epsilon/(n-1)} \right\rceil x_{ij} \end{aligned}$$

This rounding scheme increases each edge-benefit by at most $v\epsilon/(n-1)$ and each path-benefit by at most $v\epsilon$. Then, the test runs Dual-Reverse-DP over graph G with scaled edge-benefits $\lceil b_{ij}/(v\epsilon/(n-1))\rceil$. But it computes $r_j(b)$ for $b = 0, 1, \ldots$ until $r_1(b) > r$ for some $b = b' < (n-1)/\epsilon$, or else $b = \hat{b} \geq (n-1)/\epsilon$. In the first case,

$$b' < \frac{n-1}{\epsilon} \Leftrightarrow \frac{v\epsilon}{n-1} b' < \frac{v\epsilon}{n-1}\frac{n-1}{\epsilon} \Rightarrow b^\star < v$$

meaning that every r-path has benefit at most $b^\star < v$. In the second case,

$$\hat{b} \geq \frac{n-1}{\epsilon} \Leftrightarrow \frac{v\epsilon}{n-1}\hat{b} \geq \frac{v\epsilon}{n-1}\frac{n-1}{\epsilon} \Leftrightarrow \frac{v\epsilon}{n-1}\hat{b} - v\epsilon \geq v - v\epsilon \Rightarrow b^\star \geq v(1-\epsilon)$$

meaning that an r-path of benefit at least $v(1-\epsilon)$ has been found. The test is formalized as follows.

Test(v)

```
b ← 0
for (i, j) ∈ E do
    b_ij ← ⌈ min{b_ij, v} / (vε/(n−1)) ⌉
end for
if b ≥ (n − 1)/ε then
    Output "no"
end if
Use Dual-Reverse-DP to compute r_j(b) for j = 1, ..., n
```

9: **if** $r_1(b) > r$ **then**
10: Output *"yes"*
11: **else**
12: $b \leftarrow b + 1$
13: Repeat from step 5
14: **end if**

The steps 1 through 4 of Test(v) run in time $O(|E|\log(n/\epsilon))$ because the rounding up in step 3 can be performed in time $O(\log(n-1)/\epsilon)$ with binary search. Each iteration of Dual-Reverse-DP runs in time $O(|E|)$. Because there are $O(n/\epsilon)$ iterations, steps 5 through 14 run in time $O(|E|n/\epsilon)$. The running time of Test(v) is therefore $O(|E|n/\epsilon)$.

The approximate algorithm requires a lower bound $\underline{b}$ and an upper bound $\bar{b}$ for $b^\star$ to apply Test(v). As discussed above, a straightforward upper bound is $\bar{b} = (n-1)\max\{b_{ij} : (i,j) \in E\}$. A tighter bound consists of the length of the longest, resource-unconstrained path from source to sink which is obtained in $O(|E|)$ time. A simple lower bound is $\underline{b} = \min\{b_{ij} : (i,j) \in E\}$. A better lower bound consists of the benefit of the shortest, resource-unconstrained path from source to sink which is computed in $O(|E|)$ time.

A lower bound $\underline{b}$ is an ϵ-approximation to $b^\star$ if $\underline{b} \geq \bar{b}(1-\epsilon)$. Otherwise Test($v$) is applied to improve the bound with $\underline{b} < v < \bar{b}(1-\epsilon)$. If Test($v$)= *"yes"*, then the upper bound $\bar{b}$ is reduced to v, otherwise the lower bound $\underline{b}$ is increased to $v(1-\epsilon)$. The test is repeated until the ratio $\bar{b}/\underline{b}$ gets below a constant, say φ, at which point Dual-Reverse-DP is applied with scaled edge-benefits $\lceil b_{ij}/(\underline{b}\epsilon/(n-1))\rceil$ to produce an ϵ-approximation.

The ratio $\bar{b}/\underline{b}$ is best reduced via a binary search on the interval $(\underline{b}, \bar{b})$ in logarithmic scale. If the test value for the binary search is v, then the ratio either becomes $v/\underline{b}$ or $\bar{b}/v$. The size of the resulting interval should be the same to maximize the ratio reduction in the worst case, meaning that $v/\underline{b} = \bar{b}/v$ and therefore $v = \sqrt{\underline{b}\bar{b}}$.

The running time of the ϵ-approximation algorithm is a direct function of the number of steps to reduce $\bar{b}/\underline{b}$ below the desired ratio φ and the complexity of Test(v). Since $v = \sqrt{\underline{b}\bar{b}}$, we assume in the worst-case scenario that Test(v) always returns *"yes"* and the ratio is normalized with $\underline{b} = 1$ for the purpose of analysis. Thus, the number k of steps to reach the ratio φ is given by $\sqrt{\cdots\sqrt{\sqrt{\bar{b}/\underline{b}}}} \leq \varphi \Leftrightarrow (\bar{b}/\underline{b})^{(\frac{1}{2})^k} \leq \varphi \Leftrightarrow \log_\varphi(\bar{b}/\underline{b})^{(\frac{1}{2})^k} \leq \log_\varphi \varphi \Leftrightarrow 2^k \geq \log_\varphi(\bar{b}/\underline{b}) \Leftrightarrow k \geq \log_2 \log_\varphi(\bar{b}/\underline{b})$. The ϵ-approximation algorithm is detailed below.

FPTAS$(\underline{b}, \bar{b}, \varphi, \epsilon)$
while $\bar{b}/\underline{b} > \varphi$ **do**
  $v = \sqrt{\underline{b}\bar{b}}$
  if Test(v)= *"yes"* **then**
    $\bar{b} \leftarrow v$
  else
    $\underline{b} \leftarrow v(1-\epsilon)$
  end if
end while
Set $b_{ij} \leftarrow \lceil b_{ij}/(\underline{b}\epsilon/(n-1)) \rceil$
Apply Dual-Reverse-DP to obtain an optimal r-path

The algorithm reaches the desired ratio in $O(\log\log(\bar{b}/\underline{b}))$ steps. Each step requires $O(|E|n/\epsilon)$ time to run Test(v) and $O(\log\log(\bar{b}/\underline{b}))$ time to compute the test point. Thus, the ϵ-approximation algorithm runs in $O\left(\log\log(\bar{b}/\underline{b})(|E|n/\epsilon + \log\log(\bar{b}/\underline{b}))\right)$ time.

3.2.1 Example

The ϵ-approximation algorithm was applied to the sample instance with the aim of illustrating its behavior. With lower bound $\underline{b} = 1$, upper bound $\bar{b} = 20$, and $\epsilon = 0.01$, the algorithm produced the results given in Tab. 5 for $\varphi \in \{1.03, 1.2, 2\}$. For all these φ the algorithm outputs path $\langle 1, 2, 4, 6, 7, 8, 10 \rangle$ with resource consumption 10 and benefit 14, which is an optimal path from source to sink.

Table 5: Approximation algorithm applied to the sample instance.

	$\varphi = 2$			$\varphi = 1.2$			$\varphi = 1.03$		
k	$\underline{b}^k$	$\bar{b}^k$	$\bar{b}^k/\underline{b}^k$	$\underline{b}^k$	$\bar{b}^k$	$\bar{b}^k/\underline{b}^k$	$\underline{b}^k$	$\bar{b}^k$	$\bar{b}^k/\underline{b}^k$
1	1.0000	20.0000	20.0000	1.0000	20.0000	20.0000	1.0000	20.0000	20.0000
2	4.4274	20.0000	4.5173	4.4274	20.0000	4.5173	4.4274	20.0000	4.5173
3	9.3159	20.0000	2.1469	9.3159	20.0000	2.1469	9.3159	20.0000	2.1469
4	13.5133	20.0000	1.4800	13.5133	20.0000	1.4800	13.5133	20.0000	1.4800
5				13.5133	16.4398	1.2166	13.5133	16.4398	1.2166
6				13.5133	14.9049	1.1030	13.5133	14.9049	1.1030
7							13.5133	14.1921	1.0502
8							13.7101	14.1921	1.0352
9							13.8095	14.1921	1.0277

By varying the parameter ϵ, from 0.01 through 0.07, and the performance ratio $\varphi \in \{1.1, 1.2, 1.5, 2\}$, the ϵ-approximation algorithm solved 100 times the sample instance for each parameter set-up. The averages over the running times taken by the algorithm appear in Fig. 7. The algorithm was implemented in Matlab and ran on a laptop with a 2.0GHz Intel Pentium Processor and GNU Linux. As should be expected, the plot clearly shows a decrease in computational time as the performance ratio φ and approximation parameter ϵ increase.

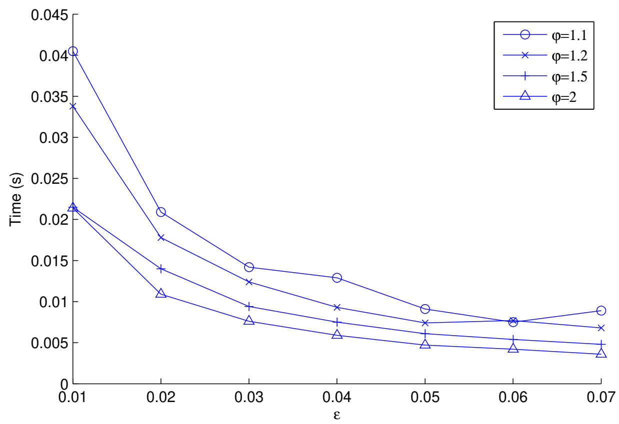

Figure 7: Influence of φ and ϵ on the approximation algorithm applied to the sample instance.

4 Computational Experiments

This section reports computational results from the application of the dynamic programming and approximation algorithm to instances of CLPP. The experiments aim to assess the relative performance between these algorithms. A pseudo-random procedure was developed to generate acyclic graphs, whose parameters include the number of nodes, the average number of incident arcs in each node, and bounds for benefit and resource consumption of the arcs. Three instances were synthesized, namely instance I_1, I_2, and I_3. Properties and other parameters of these instance appear in Table 6. The table gives the least and maximum benefit for paths connecting source s to sink t. For instance I_1, this means that $\underline{b} = 71,384$ is a lower bound for the objective function, whereas $\bar{b} = 563,284$ is an upper bound. The table also gives the least and maximum resource consumption for $s-t$ paths and the resource available (r). These bounds are easily obtained by computing unconstrained shortest and longest paths.

Table 6: Properties and parameters of the instances

Parameters	Instances I_1	I_2	I_3
Nodes	77	141	176
Arcs	411	775	916
Least benefit	71,384	76,999	138,346
Maximum benefit	563,284	981,224	1,145,516
Least cost	46,776	87,412	104,051
Maximum cost	545,096	1,012,884	1,160,784
Resource available	235,994	408,014	475,066

The dynamic programming and approximation algorithms were implemented in C language, in a workstation running the operating system GNU Ubuntu and equipped with a 2.00GHz Intel Core-Duo Processor and having 2Gb of RAM. The benefit, resource consumption, and computational time necessary to obtain the resource constrained longest path with dynamic programming are reported in Table 7.

Table 7: Results obtained with dynamic programming

Instance	Benefit	Resources	CPU (s)
I_1	536,723	167,903	7.34
I_2	957,465	405,322	24.98
I_3	1,107,836	460,333	36.22

The FPTAS was applied to each of the instances for $\varphi \in \{1.1,\ 1.5,\ 2.0\}$ and $\epsilon \in \{0.01,\ 0.02,\ 0.03,\ 0.04,\ 0.05\}$. The initial lower bound was set at $\underline{b}^{(0)} = 1$, whereas the initial upper bound was set at $\bar{b}^{(0)} = 2,000,000$ for all the instances. Tables 8, 9, and 10 show the results for I_1, I_2, and I_3 for $\varphi = 1.1$ and varying ϵ, respectively. These tables give the final lower and upper bounds, the ratio between these bounds, the number of iterations, the total benefit and resource consumption of the best path, and the CPU time taken by the approximation algorithm. Notice that the ratio $\bar{b}^\star/\underline{b}^\star$ between the final upper and lower bound is not greater than $\varphi = 1.1$. Similar results were obtained for $\varphi = 1.5$ and $\varphi = 2.0$. From these experiments, some remarks are made:

1. the approximation scheme produces optimal solutions even though the distance between upper and lower bound is about 10%.

2. the approximation scheme is substantially faster than the DP algorithm, consuming 11% of the time taken by DP on average when $\epsilon = 0.01$ and only 2% when $\epsilon = 0.05$.

Table 8: FPTAS applied to instance I_1 with $\varphi = 1.1$

	Final bounds				Solution		
ϵ	$\underline{b}^\star$	$\overline{b}^\star$	$\overline{b}^\star/\underline{b}^\star$	Iter	Benefit	Resources	CPU (s)
0.01	526,317.4375	563,650.9375	1.07	8	536,723	167,903	0.71
0.02	509,841.0937	552,584.5000	1.08	8	536,723	167,903	0.35
0.03	493,719.4062	541,625.1250	1.09	8	536,723	167,903	0.23
0.04	509,541.9687	554,425.7500	1.08	8	536,723	167,903	0.17
0.05	494,026.5000	541,383.0000	1.09	8	536,723	167,903	0.13

Table 9: FPTAS applied to instance I_2 with $\varphi = 1.1$

	Final bounds				Solution		
ϵ	$\underline{b}^\star$	$\overline{b}^\star$	$\overline{b}^\star/\underline{b}^\star$	Iter	Benefit	Resources	CPU (s)
0.01	928,665.1250	998,921.1875	1.08	8	957,465	405,322	2.83
0.02	900,592.6875	984,761.1250	1.09	8	957,465	405,322	1.36
0.03	892,967.5625	970,659.5625	1.09	9	957,465	405,322	0.99
0.04	918,352.4375	986,580.5625	1.07	8	957,465	405,322	0.65
0.05	895,502.2500	965,738.8125	1.08	8	957,465	405,322	0.50

The influence of the parameters φ and ϵ on the approximate solution to instances I_1, I_2, and I_3 are depicted in Figures 8, 9, and 10 respectively. These plots illustrate the increase in computational time with the decrease in the ratio $\varphi = \overline{b}/\underline{b}$ between upper and lower bound. They also indicate that the computational time decreases with the increase of ϵ within the range of consideration.

All in all, the computational results show that the fully polynomial approximation scheme can yield an optimal solution to the resource constrained longest-path problem far more quickly than dynamic programming.

Table 10: FPTAS applied to instance I_3 with $\varphi = 1.1$

	Final bounds				Solution		
ϵ	$\underline{b}^\star$	$\overline{b}^\star$	$\overline{b}^\star/\underline{b}^\star$	Iter	Benefit	Resources	CPU (s)
0.01	1,039,962.5625	1,115,831.5000	1.07	8	1,107,836	460,333	4.57
0.02	1,075,110.7500	1,157,911.1250	1.08	8	1,107,836	460,333	2.11
0.03	1,046,048.2500	1,136,675.5000	1.09	8	1,107,836	460,333	1.42
0.04	1,017,482.5000	1,115,615.5000	1.10	8	1,107,836	460,333	1.00
0.05	1,039,994.6875	1,122,367.1250	1.08	9	1,107,836	460,333	0.91

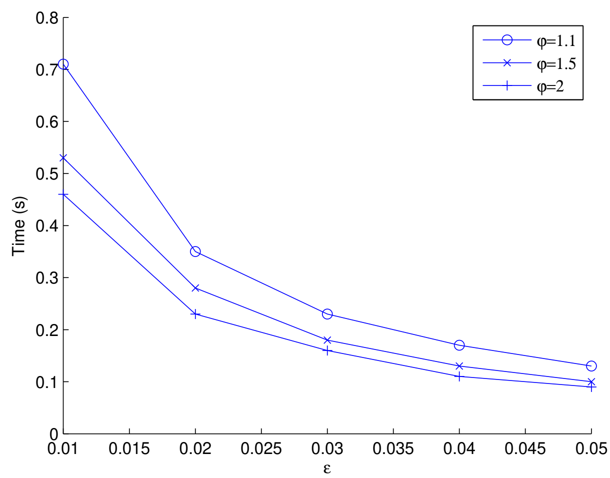

Figure 8: The influence of φ and ϵ on the approximate solution to instance I_1.

5 Extensions

This section presents alternative forms for the resource constrained longest-path problem that have a non-additive objective, non-additive resource constraints, and probabilistic or chance constraints. Models and algorithm outlines are given for these extensions.

5.1 Non-Additive Objectives

Take the problem of finding an r-path that maximizes the minimum benefit of the arcs along the path. An application arises when benefits represent probabilities of success and one wants to maximize the lowest probability, which corresponds to

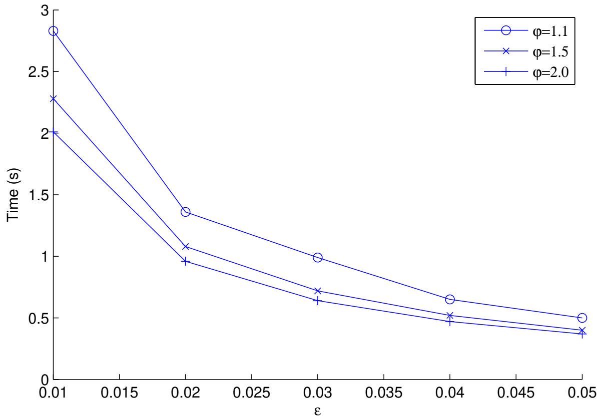

Figure 9: Influence of φ and ϵ on the approximate solution to instance I_2.

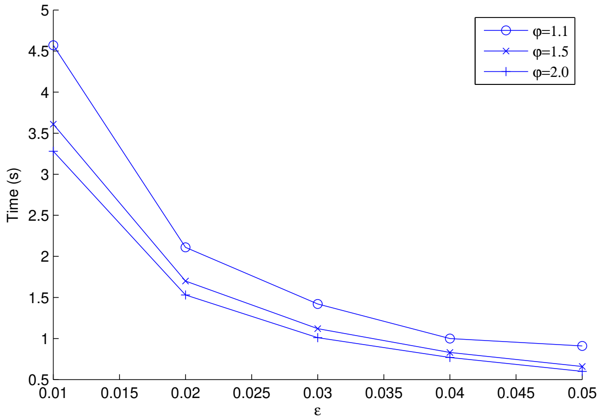

Figure 10: Influence of φ and ϵ on the approximate solution to instance I_3.

the weakest link. The original objective (2a) is additive, whereas the alternative is non-additive and given by:

$$\max_x \ \min\{b_{ij}x_{ij} : (i,j) \in E\}$$

where $x = (x_{ij} : (i,j) \in E)$ is the decision vector. A path from node i to sink t is a b-min-path if the benefit of any arc in the path is at least b. Let $r_i^{nad}(b)$ be the resource consumption of a b-min-path, from node i to sink t, of least consumption which is defined recursively by:

$$r_i^{nad}(b) = \begin{cases} \min\limits_{(i,j)\in E: b_{ij}\geq b} \{r_j^{nad}(b) + r_{ij}\}, & \text{if } \{(i,j)\in E : b_{ij} \geq b\} \neq \emptyset \\ \infty, & \text{otherwise} \end{cases} \tag{5}$$

Let $b^{\star}_{nad}$ be the maximum non-additive benefit among all r-paths from source to sink. The path that yields $b^{\star}_{nad}$ is found by setting:

$$r_i^{nad}(b) = \begin{cases} 0, & \text{if } i = t \text{ and } b = 0 \\ \infty, & \text{if } i = t \text{ and } b > 0 \\ \infty, & \text{if } i \neq t \text{ and } b > b^{\star}_{nad} \end{cases}$$

and recursively applying equation (5). The optimal solution to the non-additive objective, constrained longest-path problem is given by $r_1^{nad}(b^{\star}_{nad})$. A dynamic programming algorithm to solve recursion (5) can be devised in the same manner of the one designed for an additive objective. Further, it is relatively easy to compute an upper bound for $b^{\star}_{nad}$ for use in the DP algorithm.

5.2 Non-Additive Constraints

Another variation of the baseline problem arises when the resource is non-additive. One example is when the resource consumption of a path is given by its most consuming link—the longest edge may restrict the size of the battery of a robot, the amount of fuel between refueling stations, or the power consumption between adjacent nodes to transmit data. In such cases, constraint (2b) becomes:

$$\max\{r_{ij}x_{ij} : (i,j) \in E\} \leq r$$

The problem asks to find a path from source s to sink t of maximum benefit such that each arc on the path consumes at most r units. This problem can be reduced to a resource-unconstrained problem by removing every arc (i,j) for which $r_{ij} > r$ and then solving a standard longest-path problem in an acyclic graph. The solution is then found in time $O(|E|)$.

5.3 Probabilistic Constraints

Consider a scenario where the resource consumption in a link (i,j) is given by a random variable $\mathbf{r}_{ij}$. Rather than using the worst-case travel time from node i to j, one may choose to model travel time as a random variable. To deal with random variables, constraint (2b) has to be cast in a deterministic equivalent form. One way consists of using probabilistic or chance constraints

[Charnes and Cooper 1963, Birge and Louveaux 1997], expressed as:

$$P\left(\sum_{(i,j)\in E}\mathbf{r}_{ij}x_{ij}\le r\right)\ge\alpha\Leftrightarrow P\left(r\ge\sum_{(i,j)\in E}\mathbf{r}_{ij}x_{ij}\right)\ge\alpha \tag{6a}$$

$$\Leftrightarrow P\left(r\ge\mathbf{r}\right)\ge\alpha \tag{6b}$$

where $P(\cdot)$ denotes the probability operator and $\mathbf{r}_{ij}$ are assumed normally distributed. Then $\mathbf{r}=\sum_{(i,j)\in E}\mathbf{r}_{ij}x_{ij}$ is a normally distributed random variable with mean:

$$E[\mathbf{r}]=E[\sum_{(i,j)\in E}\mathbf{r}_{ij}x_{ij}]=\sum_{(i,j)\in E}E[\mathbf{r}_{ij}]x_{ij}$$

By assuming that $\mathbf{r}_{ij}$ are all independent, then:

$$\begin{aligned}Var\left(\mathbf{r}\right)&=\sum_{(i,j)\in E}Var\left(\mathbf{r}_{ij}x_{ij}\right)+\sum_{(i,j)\in E}\ \sum_{(l,t)\in E-\{(l,t)\}}Cov\left(\mathbf{r}_{ij}x_{ij},\mathbf{r}_{lt}x_{lt}\right)\\&=\sum_{(i,j)\in E}Var\left(\mathbf{r}_{ij}x_{ij}\right)\end{aligned}$$

where $Var(\cdot)$ is the variance operator and $Cov(\cdot,\cdot)$ gives the covariance of two random variables. Then, constraint (6b) can be expressed in the following deterministic, equivalent form:

$$\begin{aligned}r&\ge\sqrt{Var(\mathbf{r})}F_{\mathbf{x}}^{-1}(\alpha)+E[\mathbf{r}]\\&=F_{\mathbf{x}}^{-1}(\alpha)\sqrt{\sum_{(i,j)\in E}Var\left(\mathbf{r}_{ij}x_{ij}\right)}+\sum_{(i,j)\in E}E[\mathbf{r}_{ij}]x_{ij}\\&=F_{\mathbf{x}}^{-1}(\alpha)\sqrt{\sum_{(i,j)\in E}E\left[\left(\mathbf{r}_{ij}x_{ij}-E[\mathbf{r}_{ij}]x_{ij}\right)^2\right]}+\sum_{(i,j)\in E}E[\mathbf{r}_{ij}]x_{ij}\\&=F_{\mathbf{x}}^{-1}(\alpha)\sqrt{\sum_{(i,j)\in E}E\left[\left(\mathbf{r}_{ij}-E[\mathbf{r}_{ij}]\right)^2x_{ij}^2\right]}+\sum_{(i,j)\in E}E[\mathbf{r}_{ij}]x_{ij}\\&=F_{\mathbf{x}}^{-1}(\alpha)\sqrt{\sum_{(i,j)\in E}Var\left(\mathbf{r}_{ij}\right)x_{ij}^2}+\sum_{(i,j)\in E}E[\mathbf{r}_{ij}]x_{ij}\end{aligned} \tag{8}$$

where:

- $\mathbf{x}$ is a normally distributed random variable with mean $\mu=0$ and standard deviation $\sigma=1$;
- $F_{\mathbf{x}}(x)$ is the cumulative distribution[1] of random variable $\mathbf{x}$; and

[1] $F_{\mathbf{x}}(y)=P(\mathbf{x}\le y)$ is the cumulative probability distribution of $\mathbf{x}$, that is, $F_{\mathbf{x}}(y)=\int_{x=-\infty}^{x=y}f_{\mathbf{x}}(x)dx$, where $f_{\mathbf{x}}(x)$ is the probability density function.

– $F_{\mathbf{x}}^{-1}(\alpha) = \eta$ is the α-quantile, that is, $\eta = \min\{y : F_{\mathbf{x}}(y) \geq \alpha\}$.

Notice that $x_{ij}^2 = x_{ij}$ since $x_{ij} \in \{0,1\}$ and thereby inequality (8) becomes:

$$r \geq F_{\mathbf{x}}^{-1}(\alpha)\sqrt{\sum_{(i,j)\in E} Var\,(\mathbf{r}_{ij})\, x_{ij}} + \sum_{(i,j)\in E} E[\mathbf{r}_{ij}]x_{ij} \tag{9}$$

This deterministic equivalent cannot be modeled in problem P, given by (2a) through (2d), because the inequality is nonlinear. An alternative is to find an upper bound for the term $\sum_{(i,j)\in E} Var(\mathbf{r}_{ij})x_{ij}$ which would render (9) a linear inequality. One such bound is induced by a path of maximum variance from source to sink, which can be easily obtained.

It is worth remarking that the framework above can handle general distributions so long as the mean, variance, and α-quantile are provided.

6 Summary

The Internet is the world's largest repository of information on which applications in the services industry and financial markets rely, to name a few. Innumerous applications thereof access large volumes of data stored in several sites of varying reliability and availability. Rather than retrieving information and performing the computations at a single site, mobile agent technology advocates the mobility of code to more efficiently and reliably deliver the services. To complete a task, a mobile agent visits a number of hosts that provide information from sensors, retrieve data from repositories, and perform specialized services. Since the information is available at different sites and with different degrees of reliability, a mobile agent should plan an itinerary to visit the hosts which gives rise to a family of itinerary planning problems.

Heretofore, the focus has been on the computation of an itinerary to accomplish an agent's mission that maximizes overall benefit without violating a deadline. The mission is characterized by a set of resources to be collected respecting a partial order given by a UML diagram (resource diagram). As resources are available at many sites, itinerary computation consists in finding a sequence of node-resource pairs to collect the essential resources, and possibly optional ones, within the time limit while maximizing resource benefits.

This paper showed that the mobile-agent itinerary problem (MIP) can be cast as a resource constrained longest-path problem (CLPP) that captures all the elements of MIP. A dynamic-programming (DP) algorithm was developed for CLPP in acyclic graphs that runs in $\Theta(|E|\bar{b})$. This algorithm produces the longest-path from all nodes to the destination node and for all resource availabilities ranging from 0 to r (the maximum). That is, the algorithm produces a tree-family of resource-constrained longest-paths to the sink, allowing an agent to dynamically revise its itinerary as it progresses towards the sink.

Because CLPP is NP-Hard and the DP algorithm is pseudo-polynomial, an approximation algorithm was developed using the rounding technique applied to the DP algorithm. This algorithm runs in $O\left(\log\log(\bar{b}/\underline{b})(|E|n/\epsilon + \log\log(\bar{b}/\underline{b}))\right)$ time. It is said to be a fully polynomial time approximation scheme (FPTAS) because the distance to the optimal is controlled by a parameter ϵ and its running time is bounded by a polynomial on the size of the problem instance and $1/\epsilon$. The paper reports results from experiments aimed to assess the performance of the DP and approximation algorithms in a number of representative MIP instances.

The paper also discusses extensions to the models and algorithms. First, the algorithms can readily handle non-additive objectives by replacing the sum with the min operator in the recursions. Second, non-additive constraints are tackled by removing all the arcs that do not respect the bounds and then finding an unconstrained longest-path in the pruned, acyclic graph. Third, for resource consumption modeled with random variables, the paper develops a deterministic equivalent for the chance constraint.

Acknowledgments

This work was funded in part by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) under grants 479157/2006-5 and 473841/2007-0.

References

[Beasley and Christofides 1989] Beasley, J. E. and Christofides, N. : “An algorithm for the resource constrained shortest path problem”; Networks, 19 (1989) 379–394.

[Bertsekas 1995] Bertsekas, D. P.: “Dynamic Programming and Optimal Control”; Vol. I, Athena Scientific (1995).

[Birge and Louveaux 1997] Birge, J. R. and Louveaux, F.: “Introduction to Stochastic Programming”; Springer-Verlag (1997).

[Brewington et al. 1999] Brewington, B., Gray, R., Moizumi, K., Kotz, D., Cybenko, G. and Rus, D.: “Mobile agents in distributed information retrieval”; in Intelligent Information Agents, M. Klusch, Editor; Springer Verlag (1999) 355–395.

[Charnes and Cooper 1963] Charnes, A. and Cooper, W. W.: “Deterministic equivalents for optimizing and satisficing under chance constraints”; Operations Research, 11, 1 (1963) 18–39.

[Cormen et al. 2001] Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C.: “Introduction to Algorithms”; MIT Press (2001).

[Erfurth and Rossak 2003] Erfurth, C. and Rossak, W.: “Autonomous itinerary planning for mobile agents”; Proc. Symposium on Adaptative Agents and Multi-Agent Systems (April 2003) 120–125.

[Garey and Johnson 1979] Garey, M. R. and Johnson, D. S.: “Computers and Intractability: A Guide to the Theory of NP-Completeness”; W. H. Freeman and Company (1979).

[Glover and Kochenberger 2003] Glover, F. and Kochenberger, G. A., editors: “Handbook of Metaheuristics”; Kluwer Academic Publishers, Norwell, MA (2003).

[Handler and Zang 1980] Handler, G. Y. and Zang, I.: "A dual algorithm for the constrained shortest path problem"; Networks, 10 (1980) 293-310.
[Hassin 1992] Hassin, R.: "Approximation schemes for the restricted shortest path problem"; Mathematics of Operations Research, 17, 1 (1992) 36–42.
[Joksch 1966] Joksch, H. C.: "The shortest route problem with constraints"; Journal of Mathematical Analysis and Applications, 14, (1966) 191–197.
[Kotz and Gray 1999] Kotz, D. and Gray, R. S.: "Mobile agents and the future of Internet"; ACM Operating Systems Review, 33, 3 (May 1999) 7–13.
[Lange and Oshima 1998] Lange, D. B. and Oshima, M.: "Programming and Deploying Java Mobile Agents with Aglets"; Addison-Wesley Longman, Reading, MA (1998).
[Lange and Oshima 1999] Lange, D. B. and Oshima, M.: "Seven good reasons for mobile agents"; Communications of the ACM, 42, 3 (March 1999) 88–89.
[Mehlhorn and Ziegelmann 2000] Mehlhorn, K. and Ziegelmann, M.: "Resource constrained shortest paths"; Proceedings of the Annual European Symposium on Algorithms in Saarbrücken, Germany, LNCS 1879, Springer: London, UK (2000).
[Shima 2006] Shima, R. B.: "Constrained Shortest Paths: A Review and Applications"; Master's Dissertation, Graduate Program in Electrical Engineering, Federal University of Santa Catarina, Florianópolis, Brazil, in Portuguese (June 2006).
[Rech et al. 2005] Rech, L., Oliveira, R. S. and Montez, C. B.: "Dynamic determination of the itinerary of mobile agents with timing constraints"; Proc. IEEE/WIC/ACM International Conference on Intelligent Agent Technology, Compiègne, France (2005) 45–50.
[Rech et al. 2006] Rech, L., Montez, C. B. and Oliveira, R. S.: "A clone-pair approach for the determination of the itinerary of imprecise mobile agents with firm deadlines"; Proc. 11th IEEE International Conference on Emerging Technologies and Factory Automation, Prague, Czech Republic (2006).
[Rech 2006] Rech, L.: "Dynamic Determination of the Itinerary of Mobile Agents with Timing Constraints"; Ph.D. Dissertation, Graduate Program in Electrical Engineering, Federal University of Santa Catarina, Florianópolis, Brazil, in Portuguese (2006).
[Sahni 1977] Sahni, S.: "General techniques for combinatorial approximation"; Operations Research, 25 (1977) 920–936.
[Tolbert et al. 2001] Tolbert, L. M., Qi, H. and Peng, F. Z.: "Scalable multi-agent system for real-time electric power management"; Proc. IEEE Power Engineering Society Summer Meeting, Vancouver, Canada (June 2001) 1676–1679.
[Wu et al. 2004] Wu, Q., Rao, N. S. V., Barhen, J., Iyengar, S. S., Vaishnavi, V. K., Qi, H. and Chakrabarty, K.: "On computing mobile agent routes for data fusion in distributed sensor networks"; IEEE Transactions on Knowledge and Data Engineering, 16, 6 (June 2004) 740–753.
[Xiao et al. 2005] Xiao, Y., Thulasiraman, K., Xue, G., and Jüttner, A.: "The constrained shortest path problem: algorithmic approaches and an algebraic study with generalization"; AKCE Int. Journal of Graphs and Combinatorics, 2, 2 (2005) 63–86.
[Ziegelmann 2001] Ziegelmann, M.: "Constrained Shortest Paths and Related Problems"; Ph.D. Dissertation, Universität des Saarlandes, Germany (July 2001).