The Travelling Agent Problem

Katsuhiro Moizumi (*katsuhiro.moizumi@dartmouth.edu*)
George Cybenko, Corresponding author (*gvc@dartmouth.edu*)
Thayer School of Engineering, Dartmouth College, Hanover NH 03755 USA
Voice: (603) 646-3843 FAX: (603) 646-2277

February 4, 1998

Abstract

This paper considers a sequencing problem which arises naturally in the scheduling of software agents. We are given n sites at which a certain task might be successfully performed. The probability of success is p_i at the ith site and these probabilities are independent. Visiting site i and trying the task there requires time (or some other cost metric) t_i whether successful or not. Latencies between sites i and j are l_{ij}, that is, the travel time between those two sites. Should the task be successfully completed at a site then any remaining sites do not need to be visited. The *Travelling Agent Problem* is to find the sequence which minimizes the expected time to complete the task. The general formulation of this problem is NP-Complete. However, if the latencies are constant we show that the problem can be solved in polynomial time by sorting the ratios p_i/t_i according to decreasing value and visiting the sites in that order. This result then leads to an efficient algorithm when groups of sites form *subnets* in which latencies within a subnet are constant but can vary across subnets. We also study the case when there are deadlines for solving the problem in which case the goal is to maximize probability of success subject to satisfying the deadlines. Applications to mobile and intelligent agents are described.

Keywords: Stochastic control, mobile agents, intelligent agents, planning, dynamic programming.

Copyright 1998 by Katsuhiro Moizumi and George Cybenko.
Submitted to "Mathematics of Control, Signals and Systems," January 1998.
Available at <http://agent.cs.dartmouth.edu/papers/moizumi:agent-plan.ps.Z>.

1 Introduction

Suppose you are shopping for a specific item known to be sold in $n+1$ stores, s_i where $0 \leq i \leq n$. The probability that store i has the item is known and given by p_i. Moreover, it takes a known time t_i to navigate through store i to the section where the item is stocked thereby determining whether the item is available or not. Going from store i to store j requires travel time l_{ij}. Starting and ending at store s_0, with $0 = t_0 = p_0$, what is the minimal expected time to find the item or conclude that it is not available?

As this shopping analogy suggests, once the item is found, you are done and can return to s_0 by the most expedient route which may or not may be taking the direct path requiring l_{i0} travel time if the item was found at s_i. With probability $\prod_{i=1}^{n}(1-p_i) = \prod_{i=0}^{n}(1-p_i)$ all sites must be visited. Moreover, once a store is visited and found not to have the item in stock, there is no reason to go back to that store. Probabilities for success at different sites are assumed independent. Site s_0 is the start and end of the shopping task and can be considered "home." Since $0 = p_0 = t_0$, s_0 only contributes to the problem through latencies.

This problem arises when planning the actions of mobile software "agents" [G, JvRS, W]. Mobile agents are programs which autonomously move within a computer network from machine to machine typically in an effort to satisfy an information retrieval and processing requirement. Using the shopping metaphor from above, the "stores" are information servers such as databases or web servers. The probabilities of success, p_i can be thought of as estimated from relevance scores given by search engines such as Altavista, Infoseek and others. Compute times, t_i, and latencies, l_{ij}, are obtained from network status monitors.

In this framework, an agent has a specific information request to satisfy, such as finding a topographical map of a given region say. The search engine or directory service identifies locations, s_i for $i = 1, ..., n$, together with probabilities (relevance scores), p_i, for finding the required data at the corresponding sites.

The agent then queries the network status monitor to find latencies and estimated compute times, l_{ij} and t_i, for those sites. Based on this information, an autonomous agent must plan its itinerary to minimize expected time for successfully completing the task. Such a mobile agent system has been developed by us at Dartmouth [G, RRK] and is known as *Agent Tcl*. See http://www.cs.dartmouth.edu/˜ agent/. In addition to this mobile agent application, there are numerous other planning and scheduling problems which can be formulated in these terms [P].

Some work on agent planning follows heuristic methods developed by the artificial intelligence community [AIS, CT, DB, FN]. Our work is devoted to finding optimal solutions to specific planning problems using efficient algorithms from combinatorial optimization, operations research and stochastic control [B57, B85, CGM, H].

Formally, we state the problem as:

> *The Travelling Agent Problem* – There are $n+1$ sites, s_i with $0 \leq i \leq n$. Each site has a known probability, $0 \leq p_i \leq 1$, of being able to successfully complete the agent's task and a time, $t_i > 0$, required for the agent to attempt the task at s_i regardless of whether successful or not. These probabilities are independent of each other. Travel times or latencies for the agent to move between sites are also known and given by $l_{ij} \geq 0$ for moving between site i and site j. When the agent's task has been successfully completed at some site, the agent must return to site 0 from which it started. For site 0, $p_0 = t_0 = 0$. The *Travelling Agent Problem* is to minimize the expected time to successfully complete the task.

Several comments are appropriate.

- The latencies, l_{ij}, can be assumed to be the minimal travel time between nodes i and j as would typically be used in network routing. This observation avoids the situation where an indirect path between nodes, without "stopping" at the sites along the indirect path, might be shorter that the direct path.

- The probabilities, p_i, can be thought of as conditioned on attempting the task at site i. That is

$$p_i = \text{Prob(success at site } i \mid \text{site } i \text{ not visited yet)}.$$

Accordingly,

$$\text{Prob(success at site } i \mid \text{site } i \text{ has been visited)} = 0 \text{ or } 1.$$

This formally handles the site revisiting issue.

- This problem can be formulated as a Markov Decision Problem or discrete stochastic control problem [B85, H] in which the state space consists of vectors indexed by sites with coordinate values indicating whether a site has been visited already or not. Standard dynamic programming algorithms could be used on this formulation but since the state space is exponentially large in the number of sites, this formulation is not scalable. However, we will see that in certain cases, the state space can be simplified leading to efficient dynamic programming solutions.

- If all sites must be visited (so that p_i is irrelevant), then the problem reduces to the classical Travelling Salesman Problem (TSP) which is known to be NP-Complete.

Section 2 contains the main body of technical results, namely efficient algorithms for restricted instances of the problem. Extensions of results from Section 3 to the so-called *multiple subnetwork* case using dynamic programming are given. Section 4 deals with deadlines while Section 5 is the conclusion and summary.

2 The Travelling Agent Problem

The formal definition of the Travelling Agent Problem has been described in the previous section. A solution to the problem consists of specifying the order in which to visit the sites, namely a permutation $< i_1, i_2, ..., i_n >$ of 1 through n. Such a permutation will be called a *tour* in keeping with tradition for such problems.

The expected time to complete the task or visit all sites in failure for a tour $T =< i_1, i_2, ..., i_n >$ is

$$C_T = l_{0i_1} + t_{i_1} + p_{i_1} l_{i_1 0} + \sum_{k=2}^{n} \left\{ (\prod_{j=1}^{j=k-1} (1 - p_{i_j})) \right\} (l_{i_{k-1} i_k} + t_{i_k} + p_{i_k} l_{i_k 0}) + \prod_{j=1}^{n} (1 - p_j) l_{n0}. \tag{1}$$

This formula can be understood as follows. The first site, s_{i_1}, on the tour is always visited and requires travel time l_{0i_1} to reach. Upon arrival, time t_{i_1} must be spent there regardless of success or failure. With probability p_{i_1} the task is successfully completed in which case the agent can return to site 0 with time cost $l_{i_1 0}$. However, with probability $(1 - p_{i_1})$ there was failure and the agent proceeds to site i_2. The expected value of the contribution involving moving from site i_1 to site i_2 and succeeding there is

$$(1 - p_{i_1})(l_{i_1 i_2} + t_{i_2} + p_{i_2} l_{i_2 0}).$$

Here the factor $(1 - p_{i_2})$ is the probability of failing at site i_2. The third term comes from failing at sites i_1 and i_2 so has probability $(1 - p_{i_1})(1 - p_{i_2})$ which is multiplied by the expected time for success at site i_3. The general term then has the form:

$$\text{(probability of failure at the first } k-1 \text{ sites)} \times \text{(expected time for success at site } i_k).$$

Finally, the last term arises when failure occurs at all nodes and we must return to the originating site. We have used independence of the various probabilities here. Not surprisingly, this problem is NP-complete [K] as will be shown below. Note that in the proof, the question is altered so that we are asking whether there is a tour whose cost, as above, is smaller than or equal to some total length B. This can be used in a binary search method to find the minimum.

Theorem 1 *The Travelling Agent Problem (TAP) is NP-Complete.*

Proof – We start by showing that TAP belongs to NP. Given a tour, T, we can verify if the total expected length C_T is smaller than or equal to B by merely using the formula. This verification can clearly be performed in polynomial time (O(n^2) steps specifically). Thus, TAP belongs to NP.

Next, we show that the problem is NP-Hard, by proving that the Hamiltonian Cycle Problem [GJ] can be reduced to TAP. A Hamiltonian Cycle in graph $G =< V, E >$ is a simple circuit that includes all the vertices V. Thus, a cycle is expressed as an ordering of the vertices $< v_1, v_2, ..., v_k >$ such that $\{v_i, v_{i+1}\} \in E$ for $1 \leq i \leq k$ and $\{v_k, v_1\} \in E$.

Define a TAP with probabilities strictly between 0 and 1 and

$$l_{ij} + t_j = \begin{cases} 0 & \text{if} \quad i, j \in E \\ 1 & \text{if} \quad i, j \notin E, \end{cases}$$

so that $t_j = 0$ for any vertex on an edge in E and $l_{ij} = 0$ for any edge in E. This formation can be done in polynomial time.

The graph G has a Hamiltonian cycle if and only if the corresponding TAP has a tour with expected cost of 0. To see this, assume that the graph G has a Hamiltonian cycle H. The corresponding tour T in the TAP will have cost 0 because all the time costs for T are 0. On the other hand, if the tour T has cost 0, the latencies and site times must all be 0 along this tour by construction. Here we use the fact that the probabilities are strictly between 0 and 1 so that the only way for the cost to be 0 is for the times to be 0 along the tour. Thus graph G has a Hamiltonian cycle since all the edges in the tour T have to belong to E by construction again. Q.E.D.

2.1 Constant Latencies

The complexity of the problem can be reduced when latencies between nodes are equal. For example, if the processing time at each node is extremely large (compared to the latency between the nodes), differences among the latencies could be ignored, or even taken to be zero. Alternately, if no information about internodal latencies are known, we might assume all of them to be constant. The constant latency assumption is reasonable in the case of a single subnetwork as well. Accordingly, the assumption that we employ in this section is:

Assumption 1 *Latencies between nodes are all the same.*

Under this assumption, it turns out that the TAP can be solved in polynomial time, as we will see below.

Theorem 2 *Assume in the TAP that $l = l_{ij} = l_{km} \geq 0$ for all i, j, k, m, namely Assumption 1. Compute times, t_i, and probabilities, p_i, can be arbitrary. Then the optimal tour for the TAP is attained if the nodes are visited in decreasing order of $p_i/(t_i + l)$.*

Proof – The proof uses an interchange argument commonly used in finance and economics. See [B87] for example. For notational convenience and without loss of generality consider the specific tour $T =< 1, 2, 3, ..., n >$. The total expected cost for the tour T is as in equation (1) with notational changes:

$$\begin{aligned} C_T &= l_{01} + t_1 + p_1 l_{10} + \sum_{k=2}^{n} \{(\prod_{j=1}^{k-1}(1-p_j))\}(l_{k-1k} + t_k + p_k l_{k0}) + \prod_{j=1}^{n}(1-p_j) l_{n0} \\ &= l + t_1 + p_1 l + \sum_{k=2}^{n} \{(\prod_{j=1}^{k-1}(1-p_j))\}(l + t_k + p_k l) + \prod_{j=1}^{n}(1-p_j) l \\ &= 2l + t_1 + \sum_{k=2}^{n} \{\prod_{j=1}^{k-1}(1-p_j)\}(t_j + l) \end{aligned}$$

where we have used the fact that

$$p_k \cdot l \cdot \prod_{j=1}^{k-1}(1-p_j) + l \cdot \prod_{j=1}^{k}(1-p_j) = l \cdot \prod_{j=1}^{k-1}(1-p_j).$$

This also eliminates the last term so note that p_n does not explicitly arise in the final expression because regardless of the value of p_n, we return to s_0 after visiting s_n.

Now consider the effect of switching the order of two adjacent sites on the tour, say i and $i+1$ for some $1 \leq i \leq n-1$. Call this new tour, T'.

In the above expression for the expected cost, only the ith and $(i+1)$st terms are affected by the switch. The terms appearing before the ith term do not contain anything which involves i or $i+1$. On the other hand, terms that follow the $(i+1)$st term all contain $(1-p_i)\cdot(1-p_{i+1})$ in precisely the same way but no other ingredients that depend on either i or $i+1$. Note that for $i+1=n$ there are no terms following the two terms we are isolating so we can handle that case as well by the following argument.

The difference in expected cost between T and T' can be calculated by comparing only these two terms. The difference in the expected cost is therefore:

$$\begin{aligned}
C_T - C_{T'} &= (t_i+l)\prod_{j=1}^{i-1}(1-p_j) + (t_{i+1}+l)\prod_{j=1}^{i}(1-p_j) \\
&\quad -(t_{i+1}+l)\prod_{j=1}^{i-1}(1-p_j) - (t_i+l)(1-p_{i+1})\prod_{j=1}^{i-1}(1-p_j) \\
&= (t_i + l + (t_{i+1}+l)(1-p_i) - t_{i+1} - l - (t_i+l)(1-p_{i+1}))\prod_{j=1}^{i-1}(1-p_j) \\
&= (p_{i+1}(t_i+l) - p_i(t_{i+1}+l))\prod_{j=1}^{i-1}(1-p_j).
\end{aligned}$$

Thus, T is a better tour (has smaller expected cost) if

$$p_{i+1}(t_i+l) < p_i(t_{i+1}+l)$$

or equivalently,

$$\frac{p_i}{t_i+l} > \frac{p_{i+1}}{t_{i+1}+l}.$$

This shows that when two adjacent sites have the above ratios out of order (that is the ith site on the tour has a smaller ratio than the $(i+1)$st site), then we can decrease the expected cost by switching them.

So we can for example perform a Bubble Sort on any initial tour ordering and every swap in the Bubble Sort will decrease the expected time for the tour. The optimal value is then the sequence with decreasing ratios as above. Q.E.D.

Since it is possible for some of the p_i and/or t_i+l to be zero, we should handle that case as well. Clearly, any site for which $p_i = 0$ should not be visited at all, that is, it should be dropped from any prospective tour. Moreover, it might be that $t_i + l = 0$ for some of the remaining sites. In that case, we can modify the final steps of the above proof so that we visit the sites in order of increasing $(t_i+l)/p_i$ since $p_i \neq 0$.

Theorem 3 *Suppose that all latencies are that same and that some $p_i = 0$ and $t_i + l = 0$. Then the optimal tour consists of:*

- *First dropping sites with $p_i = 0$;*

- *Secondly, sorting the ratios for the remaining sites,*

$$\frac{t_i + l}{p_i},$$

into increasing order and visiting the sites in that order.

An important observation is the fact that $t_i + l$ is the expected time to reach site i and process there after failing at site $i-1$. We would obtain the same results if we used the expected time after site i but before reaching sites $i+1$ or the home sites, s_0. That expected time is

$$t_i + (1 - p_i)l + p_i l$$

which is equal to $t_i + l$ and does not change the result of above.

In fact, the expected time, $t_i + l$ can be replaced by any expected time which is *independent* of the position of the site within the tour. This will be of key importance in the next section.

This observation and a small construction allows us to solve the following modified problem.

Theorem 4 *Suppose the latency from the home site, s_0, to all the other sites is a constant, $l' \neq l$, where l is the latency between sites i and j for $1 \leq i, j \leq n$. Then the optimal tour is still obtained by:*

- *Dropping sites with $p_i = 0$;*
- *Sorting the sites into increasing order of*

$$\frac{t_i + l}{p_i}.$$

Proof – Create a new site, s^*, whose latency to all sites s_i for $i > 0$ is l and whose latency to site s_0 is $l' - l$ (this might be negative but it does not affect the argument). Consider any tour, T, using this site, s^*, as the home. Call the cost of the tour using s^* as home, D_T. Let the cost of the tour for the original problem be C_T.

Then the relationship between the costs of the tours is

$$D_T = C_T - 2(l' - l)$$

because any tour must start and end at either s_0 or s^*, even if $l' - l < 0$. Because of this relationship between costs, the minimal expected time tour for the two problems are the same. The best tour using s^* can be computed using Theorem 1 and by the above is the best tour for this modified problem as well. Q.E.D.

We will use a simple technical lemma based on the proof of Theorem 1. For this lemma, suppose that we have a general TAP with arbitrary latencies. This means that the expected time to visit site s_i, where expected time is measured from the time an agent arrives at s_i until it either successfully finishes or travels to the next site. If site s_i is followed by site s_j, then that time is

$$t_i^* = t_i + p_i l_{i0} + (1 - p_i) l_{ij}.$$

In general, this time is variable but the lemma addresses the case when a subsequence of a tour can be rearranged without affecting these expected times.

Theorem 5 *Suppose that we have a general TAP with a tour, $T = < s_1, s_2, s_3, ..., s_n >$. Suppose that for a subsequence of the tour, $< s_i, ..., s_j >$ for $i < j$, any permutation of the sites $s_i, ..., s_j$ results in the same expected execution times, t_i^*, for each of those sites, then the optimal ordering for the subsequence (that is, the permutation of the subsequence that minimizes expected time) is obtained by the permutation $< s_{k_i}, s_{k_{i+1}}, ..., s_{k_j} >$ in which*

$$\frac{t_{k_i}^*}{p_{k_i}} \leq \frac{t_{k_{i+1}}^*}{p_{k_{i+1}}} \leq ... \leq \frac{t_{k_j}^*}{p_{k_j}}.$$

PROOF – As in the proof of Theorem 2, consider switching two adjacent sites, say $s_{k_m}, s_{k_{m+1}}$, in any ordering of the subsequence in question. Call the original tour T and the tour with switched sites T'. Since the expected times for these sites are independent of their ordering in the subsequence, we as above see that

$$C_T - C_{T'} = P(p_{k_{m+1}} t^*_{k_m} - p_{k_m} t^*_{k_{m+1}}) \leq 0$$

if and only if

$$\frac{t^*_{k_{m+1}}}{p_{k_{m+1}}} \geq \frac{t^*_{k_m}}{p_{k_m}}.$$

Thus this sorted order minimizes the expected time for the subsequence of sites. Q.E.D.

2.2 The Multiple Subnetwork Case

Assumption 1 and Theorem 2 address the case of completely constant latencies. Theorem 3 offers a small generalization in which latencies to the home node can be different but still constant. However, many situations can be modeled by variable latencies which are constant within subnetworks and across subnetworks. Specifically, consider the case of two subnetworks (one in Japan and one in the US). Latencies between any two nodes within the same subnetwork are constant as are latencies across the two subnetworks. That is, for sites in Japan, latencies are a constant, l_J, and in the USA they are l_U. Latencies between two nodes, one in Japan and one in the USA, are a third constant, l_{JU}. Formally, we define the Two Subnetwork Travelling Agent Problem (TSTAP) as follows.

Assumption 2 *The relevant sites belong to two subnetworks, S_1 and S_2. Sites in S_i are s_{ij} where $1 \leq j \leq n_i$. There are three latencies: $L_1, L_2, L_{12} \geq 0$. For $s_{1j} \in S_1, s_{2k} \in S_2$, $l_{1j2k} = l_{2k1j} = L_{12}$ while for $s_{1j}, s_{1k} \in S_1$, we have $l_{1j1k} = l_{1k1j} = L_1$. Similarly, for $s_{2j}, s_{2k} \in S_2$, we have $l_{2j2k} = l_{2k2j} = L_2$. Probabilities, $p_{mj} > 0$ are nonzero and independent as before. Compute times $t_{mj} \geq 0$ are arbitrary but nonnegative. Latencies between the home site, s_0, and sites in S_i are L_{0i}. We assume that $L_{0i}, L_{i2} \geq L_i$. That is, latencies within a subnetwork are smaller than latencies across networks and to the home sites.*

Under Assumption 2, the Two Subnetwork Travelling Agent Problem (TSTAP) can be solved in polynomial time using the results of Theorem 4 and dynamic programming. The result can be generalized in several ways but we defer that discussion until after the basic case is handled. Formally, we will show:

Theorem 6 *The optimal (minimal expected time) sequence for the TSTAP can be computed in polynomial time, $O(n_1 \log n_1 + n_2 \log n_2 + (n_1 + 1)(n_2 + 1))$.*

Outline of the proof – The proof consists of two steps. The major difference between this problem and the previous cases of all constant latencies is that now the optimal solution requires making choices about whether to stay in the same subnetwork or to cross over to the other subnet after each site is visited.

We first show that the order in which sites are visited in one of the subnets is given by the ordering specified by Theorems 2, 3 and 4. This greatly reduces the number of choices necessary – after visiting a site, we only need to decide which of the eligible sites, one per subnetwork at most, should be visited next. The sorting requires $O(n_1 \log n_1 + n_2 \log n_2)$ steps.

This sorted ordering is used in the second step where a dynamic programming algorithm is used to compute the optimal solution. Even though the problem is stochastic, it can be solved by a deterministic dynamic programming algorithm in roughly $O((n_1 + 1)(n_2 + 1))$ steps.

Proof – As before, eliminate all sites with $p_{ij} = 0$ since they contribute time but no possibility of solution.

ASSERTION 1 – Suppose that the optimal tour is

$$< s_{i_1 j_1}, s_{i_2 j_2}, ..., s_{i_M j_M} >$$

where $M = n_1 n_2$. Without loss of generality, let the sites in S_i be visited in this order:

$$s_{i1}, s_{i2}, s_{i3}, ..., s_{in_i}.$$

Then

$$\frac{t_{i(j-1)} + p_{i(j-1)}L_{i0} + (1-p_{i(j-1)})L_i}{p_{i(j-1)}} \leq \frac{t_{i(j)} + p_{i(j)}L_{i0} + (1-p_{i(j)})L_i}{p_{i(j)}}$$

for $1 \leq j-1 \leq n_i - 1$.

We will show the result for subnetwork 1 and then see that it applies by symmetry to subnetwork 2 as well.

First of all, note that if a tour consists of consecutive visits to sites within S_1, then those sites within S_1 must be ordered according to the claim of the theorem by the interchange argument of Theorems 2, 3 and 4. That is, switching any two adjacent sites, s_{1j} and $s_{1(j+1)}$, within S_1 (without a intervening trip to S_2) leads to a difference in the expected time that is precisely of the form seen before. This means that the ordering has to follow

$$t^*_{1j}/p_{1j} \leq t^*_{1(j+1)}/p_{1(j+1)}.$$

Notice that although the last site within S_1 before crossing over to S_2 has a latency $L_{12} > L_1$ but that latency is independent of which S_1 is the last.

The only remaining case is when two sites within S_1 are separated on a tour by a visit to some sites within S_2. This case establishes the claim of the theorem by using agregatated sites and works only for the optimal tour, not any valid tour. We will point out where optimality of the tour is used.

Define *metasites* to be aggregated sites within subnetwork 2 that are visited *between* visits to subnetwork 1. Using the above notation, suppose that between visiting s_{1j} and $s_{1(j+1)}$ we visit only sites within subnetwork 2, say $s_{2k}, ..., s_{2m}$. We will treat the subnetwork 2 sites $s_{2k}, ..., s_{2m}$ as if they were a single site.

The expected time from starting at s_{1j} and arriving at s_{2k} is technically $t_{1j} + p_{1j}L_{10} + (1-p_{1j})L_{12}$ and the probability of success is p_{1j}. However, let us define a modified site, s^*_{1j} with

$$t^*_{1j} = t_{1j} + p_{1j}L_{10} + (1-p_{1j})L_1$$

as the new expected time and probability of success $p^*_{1j} = p_{1j}$ as before.

For the *metasite* $ms_{2k:2m}$, define the expected time to be

$$\begin{aligned} t_{ms_{2k:2m}} &= L_{12} - L_1 + t_{2k} + p_{2k}L_{20} + (1-p_{2k})L_2 \\ &\quad +(1-p_{2k})(t_{2(k+1)} + p_{2(k+1)}L_{20} + (1-p_{2(k+1)})L_2 + ... + (1-p_{2m})L_{12}..)))). \end{aligned}$$

The probability of success for this metasite is

$$p_{ms_{2k:2m}} = p_{2k} + (1-p_{2k})p_{2(k+1)} + (1-p_{2k})(1-p_{2(k+1)})p_{2(k+2)} + ... \prod_{i=k}^{m-1}(1-p_{2i})p_{2m}.$$

This time is merely the expected time to start at s_{2k} and either finish successfully or travel and start at site s_{1k} together with the difference $L_{12} - L_1$ which is leftover from the s_{1j} to s_{2k} travel time that is not accounted for in t^*_{1j}.

By splitting the expected time for site s_{1j} and the metasite $ms_{2k:2m}$ in this way, we see that the expected time to complete visiting s_{1j} and $ms_{2k:2m}$ is independent of whether another S_1 node follows s_{1j} or such a metasite follows s_{1j}. Similarly, the time for $ms_{2k:2m}$ is independent of which S_1 site preceds it.

To reiterate, we have redefined the expected time for s_{1j} so that it is as if the next site were an S_1 site instead of $ms_{2k:2m}$. Moreover, the excess latency we removed from s_{1j} has been added to the time for $ms_{2k:2m}$.

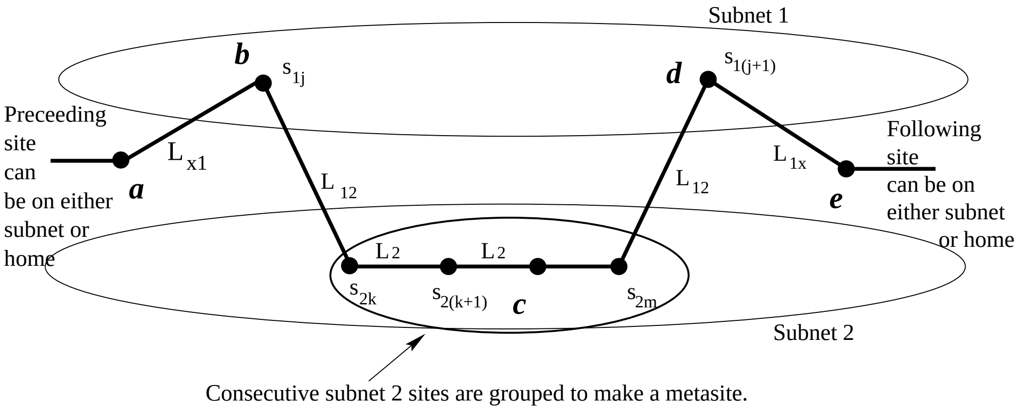

Figure 1: Definition of sites a, b, c, d, e.

This means that as long as either another S_1 site follows s^*_{1j} or $ms_{2k:2m}$ follows s^*_{1j}, the expected time for s^*_{1j} is constant. Similarly, as long as the site preceding $ms_{2k:2m}$ has a latency of L_1 to reach the site following it (that is, it is an S_1 site), the expected time for $ms_{2k:2m}$ is constant.

For notational simplicity, we will use the following names:

- The site preceding s^*_{1j} in S_1 is called a where an agent makes a decision whether it should stay in $S-1$ or move to another subnetwork;
- Sites $s_{1j},\ ms_{2k:2m},\ s_{1(j+1)}$ are called $b,\ c,\ d$ respectively;
- The site following d is called e.

This situation is depicted in Figure 1.

Now suppose that the five sites, a, b, c, d, e, form part of the optimal tour. Our goal is to show that

$$\begin{aligned} p_b/t_b &= \frac{p_{1j}}{t_{1j} + p_{1j}L_{10} + (1-p_{1j})L_1} = p_{1j}/t^*_{1j} \\ &\geq p_d/t_d = \frac{p_{1(j+1)}}{t_{1(j+1)} + p_{1(j+1)}L_{10} + (1-p_{1(j+1)})L_1} = p_{1(j+1)}/t^*_{1(j+1)}. \end{aligned}$$

We do this in two steps:

1. Showing that $p_b/t_b \geq p_c/t_c$;
2. Showing that $p_c/t_c \geq p_d/t_d$.

Step 1 – Let the cost of the optimal tour with the subsequence a, b, c, d, e be C_{abcde} and consider the cost of the tour with sites b and c switched. Let the cost of the tour with b and c switched be C_{acbde}. Then by optimality

$$\begin{aligned} 0 &\geq C_{abcdef} - C_{adcbef} \\ &= P \cdot (L_{x1} - L_{x2} + (t_b + p_bL_{10} + (1-p_b)L_{12}) + (1-p_b)(t_c + p_cL_{20} + (1-p_c)(L_{21}) \\ &\qquad -(t_c + p_cL_{20} + (1-p_c)L_{21}) - (1-p_c)(t_b + p_bL_{10} + (1-p_b)L_1)) \\ &= X \geq X - P \cdot (L_{x1} - L_{x2} + L_{21} - L_1). \end{aligned}$$

Here $P > 0$ is the probability that failure has occurred at all nodes preceding b in the tour so that we in fact visit b. L_{x1} and L_{x2} are the latencies from a to S_1 and S_2 respectively since a can be either in S_1, S_2 or s_0, the home site. The other terms arise from the parts of the tours between b and d. Note that once we arrive at d, the remaining costs are identical for both tours and therefore cancel each other in the difference. The term $L_{12} - L_1$ arises as the difference in latencies in going from a to b versus a to c.

Finally, we note that $L_{x1} - L_{x2} + L_{21} - L_1$ is 0, $2L_{12} - L_1 - L_2$ or $L_{01} - L_{02} + L_{12} - L_1 = L_{12} - L_1$ depending on whether x is 1, 2 or 0 which are all nonegative by assumptions on the latencies. This explains the last inequality.

Continuing, we have

$$\begin{aligned} 0 &\geq X - P \cdot (L_{x1} - L_{x2} + L_{12} - L_1) \\ &= P \cdot (t_b + p_b L_{10} + (1-p_b)L_1 + (1-p_b)(t_c + L_{21} - L_1 + p_c L_{20} + (1-p_c)L_{12}) \\ &\quad -(t_c + p_c L_{20} + (1-p_c)L_{21} + L_{12} - L_1) - (1-p_c)(t_b + p_b L_{10} + (1-p_b)L_1)) \\ &= P \cdot (p_c t_b - p_b t_c) \end{aligned}$$

which shows that $p_b/t_b \geq p_c/t_c$ as claimed.

Step 2 – We now switch c and d in an optimal tour as above, and with the same notational conventions, we have

$$\begin{aligned} 0 &\geq C_{abcde} - C_{abdce} = P \cdot (L_{12} - L_1 + t_c + p_c L_{20} + (1-p_c)L_{12} + (1-p_c)(t_d + p_d L_{10} + (1-p_d)L_{1x}) \\ &\quad -(t_d + p_d L_{10} + (1-p_d)L_{12}) - (1-p_d)(t_c + p_c L_{20} + (1-p_c)L_{2x} \\ &= X \geq X - P \cdot (1-p_c)(1-p_d)(L_{1x} - L_{2x} + L_{12} - L_1) \\ &= P \cdot (t_c + (1-p_c)t_d - t_d - (1-p_d)t_c) \\ &= P \cdot (p_d t_c - p_c t_d) \end{aligned}$$

so that $p_c/t_c \geq p_d/t_d$. Again, $L_{1x} - L_{2x} + L_{12} - L_1$ is nonegative as in step 1.

The two inequalities derived from Steps 1 and 2 combine to show that $p_b/t_b \geq p_d/t_d$ as claimed.

which shows $t_c/p_c \leq t_d/p_d$ is implied by The two inequalities derived from Steps 1 and 2 combine to show that if *abcde* is part of the optimal tour then $p_b/t_b \geq p_c/t_c \geq p_d/t_d$ as claimed.

> Assertion 2 – The TAP of Theorem 3 can be solved in $O((n_1+1)(n_2+1))$ time using dynamic programming after the nodes are sorted as specified in Step 1 which requires $O(n_1 \log n_1 + n_2 \log n_2)$ steps.

By Assertion 1, we can assume that the nodes have been ordered into the required sequence which dictates the order in which they are visited in each subnetwork. We define a Markov Decision Problem (MDP) with states

$$\mathcal{S} = \{(i,j,k) | 0 \leq i \leq n_1, 0 \leq j \leq n_2, k = 0 \text{ or } 1\} \cup \{F\}$$

where state (i,j,k) means that an agent has already visited sites $s_{11}, ..., s_{1i}$ and $s_{21}, ..., s_{2j}$ and is presently on subnetwork k. The terminal state is F. The states $(0,0,1)$ and $(0,0,2)$ are the same initial state.

For instance, $(0,3,2)$ means that sites s_{21}, s_{22}, s_{23} have been visited and the agent is at subnetwork 2 while no sites from subnetwork 1 have yet been visited.

In the MDP framework, we need to describe actions for each state, corresponding transition probabilities and immediate costs. For the state $(i,j,1)$ there are two possible actions: G_1 and G_2, meaning attempt to go to the next site in subnet 1 or subnet 2. For state $(i,j,1)$ and action G_1 the next state is $(i+1,j,1)$ with probability $(1-p_{1i})$ and F with probability p_{1j}. The expected immediate cost is

$$t_{1j} + p_{1j} L_{10} + (1-p_{1j}) L_1.$$

For state $(i,j,1)$ and action G_2, the next state is $(i,j+1,2)$ with probability $(1-p_{1j})$ and F with probability p_{1j}. The expected immediate cost is

$$t_{1j} + p_{1j} L_{10} + (1-p_{1j}) L_{12}.$$

The same definitions apply to states $(i, j, 2)$ with actions G_1 and G_2 as above with appropriate changes.

For states (n_1, j, k) the only allowable action is G_2 and for states (i, n_2, k) the only allowable actions are G_1. For state (n_1, n_2, k) there is only one action, to go to the terminal state F with appropriate costs.

This MDP has no cycles and so the optimal cost-to-go values can be computed using backtracking from the terminal node in time proportional to the total number of states which is $(n_1 + 1)(n_2 + 1)$. Q.E.D.

2.3 Extensions and Discussion

The above algorithm and results apply to situations with multiple subnetworks providing that internetwork latencies and latencies to the home site are larger that internetwork latencies which are constant. In that case, the algorithm requires time

$$(n_1 + 1)(n_2 + 1)...(n_m + 1)$$

for m subnetworks. In the case where $n_i = 1$, this reduces to the general case of the TAP which is *NP*-Hard and the algorithm is exponential.

We can use these results as approximation methods for general TAP's by organizing subnetworks with constant latencies that approximate the original latencies.

We have shown that the TAP is NP-complete in the general formulation. However, by clustering multiple sites and approximating latencies among them to be constant, the complexity of the problem is decreased into a polynomial time computation.

The problems dealt with in this paper assumes that (1) a single agent executes a task and that (2) network properties are static. In many cases, a task can be finished in a shorter time by multiple agents rather than a single agent. The TAP should be extended to the case where multiple agents are involved in the same task. Network properties tend not to be static in real life, and even not to be stationary. Replanning in the case of changing network statistics can be a solution to the second problem.

3 The Travelling Agent Problem with Deadlines

The above results are relevant to agent planning problems without deadline. In this section, we address the important problem of handling a deadline for each site (The Travelling Agent Problem with Deadlines). For example, agents encounter this situation when they have information about when sites are going to shutdown or be isolated due to link disconnection. The goal of the problem is to obtain tours that finish an agent's task with the highest probability of success before machines become unusable. We present a pseudo-polynomial algorithm for finding such optimal tours below.

Assumption 3 *Each site i (including the home site s_0) has a deadline, D_i, by which time an agent must visit the site or else the site becomes unavailable.*

Theorem 7 *Assume in the TAP that $l = l_{ij} = l_{km} \geq 0$ for all i, j, k, m, namely Assumption 1. Compute times, t_i, and probabilities, p_i, can be arbitrary but l and t_i are integer. The agent must finish its task by visiting sites and returning to the home site before the respective deadlines. The optimal tour for this TAP with Deadlines maximizes the probability of success without exceeding a deadline at each site. (The deadline time D_i is measured from the time the agent leaves the home site.) Such an optimal tour can be computed in pseudo-polynomial time using a dynamic programming algorithm. Input times are assumed to be integer and pseudo-polynomial means polynomial in the values of the input times, not the number of bits required to represent them.*

Proof – The proof consists of four steps. In the first step, we formalize the concept of a task's completion before deadlines. The second step is an observation that in order to avoid missing deadlines, sites should be visited in increasing order of their deadlines. The third step introduces two arrays, $e(j, s)$ and $p(j, s)$. The first array holds the boolean function which is true if there is a subset which consists of the first through jth sorted sites and which has the total time equal to s, without violating any deadlines. The second array holds the total probability of success $p(j,s)$ of the subset that makes $e(j,s) = true$. The fourth step constructs these arrays in pseudo-polynomial time using dynamic programming. The optimal tour is the sorted subset that maximizes $p(j,s)$.

Step 1 - We formalize the meaning of task's completion before deadlines in this step. We assume that the agent start time is $\tau = 0$ and that the agent must visit site i before $\tau = D_i$. If an agent visits s_i, this means that it failed at every site it visited before. The time to visit all sites before i_m in failure for a tour $T'(m) =< i_1, i_2, ..., i_m >$ where $m \leq n$ and return to the home site is:

$$
\begin{aligned}
C_{T'}(m) &= l_{01} + t_{i_1} + \sum_{k=2}^{m} (l_{i_{k-1}i_k} + t_{i_k}) \\
&= \sum_{k=1}^{m} (l + t_{i_k}) \\
&= m \cdot l + \sum_{k=1}^{m} t_{i_k}
\end{aligned}
$$

The above $C_{T'}(m)$ can be no greater than the deadline D_{i_m} at the site i_m and, in addition, the total time for all tours $T'(k)$ for $k < m$ should not exceed the deadline D_{i_k}. Although the order of visiting sites does not affects the total time $C_{T'}(m)$ and the total probability of success for the tour, the wrong order may violate the deadline constraints.

The tour which maximizes the probability of successful completion can be computed in pseudo-polynomial time using dynamic programming is shown in the following steps. Pseudo-polynomial time means that the compute time is bounded by a polynomial in the value of the inputs, not the number of bits required to represent the input.

Step 2 - Consider a tour $T =< 1, 2, ..., n >$ with sites sorted according to increasing deadlines. This tour has the minimum maximum lateness where the minimal maximal lateness for a tour $T^* =< i_1, i_2, ..., i_n >$ is defined to be $\min_{T^*}\{\max_{k=1}^{n}(C'_{T^*}(k) - D_{i_k})\}$ where $C'_{T^*}(k)$ represents the completion time at the i_kth site as above. This result is derived from Jackson's work [J] and is critical to the following property used below: *If the completion times for a sequence of sites that has increasing deadlines violate some deadline then every other ordering of that sequence will violate some deadline as well.* This follows from Jackson's result since the increasing deadline ordering minimizes the maximal lateness, which if positive must be better than the lateness of any other ordering. Therefore, all other orderings have larger lateness which by virtue of being positive means that some deadline is missed. Algorithmically, this means that we can reject a sequence of sites if they are ordered by increasing deadlines and violate some deadline – we only need to check that ordering for infeasibility of the whole set of sites.

Step 3 - We introduce two arrays of size n (number of sites) by $\min(B, D_0 - l)$ where $B = \sum_{k=1}^{n} t_k + n \cdot l$ (total possible time for visiting all sites) and $D_0 - l$ represents the deadline that an agent should leave the last site for its home machine.

For integer $1 \leq j \leq n$, let $e(j,s) = 1$ if there is a subset of $\{1, 2, .., j\}$ for which the total maximum time is exactly s and each site of which can be visited no later than its deadline. If such a subset does not exist, $e(j,s) = 0$ so e is essentially Boolean. We also define the array value $p(j,s) = 0$ if $e(j,s) = 0$ and $p(j,s) = 1 - \prod_{k \in T_{e(j,s)}} (1 - p_k)$ if $e(j,s) = 1$. Here $T_{e(j,s)}$ is the set of sites whose times add up to s (thus making the deadline constraint transparent). This subset of sites has the maximum probability of success among all such subsets and each of its sites can be visited no later than the site's deadline. That probability is precisely what is stored in $p(j,s)$.

Step 4 - The values of $e(j,s)$ and $p(j,s)$ are obtained by dynamic programming. The dimensions of the arrays e and p are both n by $\min(B, D_0 - l)$. They are initialized to be all 0.

We start the algorithm with the row $j = 1$ and proceed to calculate the following rows of e from the previous rows. For $j = 1$, $e(1, s) = 1$ if and only if either $s = 0$, or $s = l + t_1$ and $s \leq D_1$. If $e(1, s) = 0$, or $e(1, s) = 1$ and $s = 0$, $p(1, s)$ is set to be 0. If $e(j, s) = 1$ and $s = l + t_1$, $p(j, s) = p_1 = 1 - (1 - p_1)$.

For $j \geq 2$, we set the value of $e(j, s) = 1$ if and only if $e(j-1, s) = 1$ is true, $l + t_j \leq s$ and $e(j-1, s-l-t_j) = 1$ where $s \leq D_j$, or $l + t_j = s$. This means that we can construct a tour with total cost s in the case of failure in one of three possible ways:

1. There was already a tour involving sites $1, 2, ..., j - 1$ with total time s (the case $e(j - 1, s) = 1$), each site of which can be visited no later than its deadline;

2. There is a nonempty tour with total time $s - t_j - l$ involving a subset of sites $1, 2, ..., j - 1$ and we can add the site j to that tour if $s \leq D_j$ (this is the case, $e(j - 1, s - t_j - l) = 1$ and $l + t_j \leq s$ where $s \leq D_j$) ;

3. $l + t_j = s$ and so we have a tour with only the site j on it if $s \leq D_j$.

In the second case, we check that if we can add the site j to a tour by comparing s and D_j when $e(j - 1, s - l - t_j) = 1$ and $l + t_j \leq s$. It is obvious that a tour which satisfies $e(j - 1, s - l - t_j) = 1$ has a maximum lateness 0 since the tour does not violate the deadline constraints. Now if we append the site j at the end of the tour, the total time becomes s. The maximum lateness of sites in the new tour is $\max\{0, s - D_j\}$ because those sites are sorted in increasing order of their deadlines. Since the maximum lateness is minimum among those of all possible permutations of sites in the tour, it is impossible to decrease the maximum lateness by changing the order of sites in the tour. Thus, site j cannot be added to the tour that satisfies $e(j, s) = 1$ if $s > D_j$.

We also define elements of the jth row of p as follows. If $e(j, s) = 0$, $p(j, s) = 0$. We handle the cases for which $e(j, s) = 1$ in the following way:

1. There was a tour already involving a subset of $1, 2, ..., j - 1$ with a total time of s with each site of which can be visited no later than its deadline but j could not be added to any previous tour to get a new tour (case for which $e(j - 1, s) = 1$ but $e(j - 1, s - t_j - l) = 0$): $p(j, s) = p(j - 1, s)$;

2. There was no tour with total time s based on sites $1, 2, ..., j - 1$ but there is one when s_j is added to a tour (case when $e(j - 1, s) = 0$ and $e(j - 1, s - t_j - l) = 1$): $p(j, s) = p_j$ if $s \leq D_j$;

3. There was already a tour involving a subset of $1, 2, ..., j - 1$ with a total time of s and another tour with total time $s - l - t_j$ using sites $1, 2, ..., j - 1$ which can be visited no later than their respective deadlines (the case when $e(j - 1, s) = 1$, and $e(j - 1, s - t_j - l) = 1$ and $s \leq D_j$): $p(j, s) = \max\{p(j - 1, s), 1 - (1 - p_j)(1 - p(j - 1, s - l - t_j))\}$;

4. $s = l + t_j$, $e(j - 1, s) = 1$ and $s \leq D_j$ in which case $p(j, s) = \max\{p(j - 1, s), p_j\}$.

This handles all cases and clearly requires only a single scan of each row, therefore about $O(\min(D_0 - l, B))$ operations per row for n rows yielding a complexity of $O(\min(D_0 - l, B) \cdot n)$ steps..

The subset which maximizes the probability of the agent's success and whose sites can be visited no later than their respective deadlines is the one that maximizes $p(n, s)$ for $s \leq D_0 - l$ (recall that $p(j, s) = 0$ if $e(j, s) = 0$). $D_0 - l$ is the deadline by which time an agent has to leave the last site for its home site 0. By visiting all the sites in that subset in increasing order of their deadline, no exceeding of deadlines will occur. However, this order cannot guarantee the minimum expected time in visiting all the sites no later than their respective deadlines. The theorem about the optimal order (the minimum expected time) will be discussed later.

It takes at most $n \log n$ to sort the subset in increasing order of deadlines. Therefore, the optimal solution can be obtained in time that is polynomial at most in n and B, $O(2 \cdot n \cdot \min\{\sum_{k=1}^{n} t_k + n \cdot l, D_0 - l\} + n \log n)$.

The only remaining issue is that of maintaining the list of sites on a tour corresponding to $e(j,s) = 1$ and the value of $p(j,s)$ recorded in the array. This can clearly be handled within the time bounds we have demonstrated. Q.E.D.

As mentioned above, this solution for TAP with deadlines optimizes the probability of success but does not necessarily minimize the expected time to visit all sites on the tour while satisfying the respective deadlines. Note that the maximal probability tour subject to the deadline constraints may not have a minimal expected time of completion. Efficient algorithms for actually minimizing the expected time appear to be difficult to obtain and remains an open problem to our knowledge, even in the sense of pseudo-polynomial performance.

4 Conclusion

Mobile agents have received a lot of attention recently as a way to efficiently access distributed resources in a low bandwidth network. Planning is a required capability so that agents can make the best use of available resources. This paper proposes a planning problem for mobile agents that is often observed in information retrieval and data-mining applications. The problem is to find the best sequence of locations so that an agent can find desired information in minimum time if there are several possible sites that may contain the information with some probability and network statistics such as latency, bandwidth and machine load are available. We prove that the complexity of the problem is NP-complete in the general formulation but have successfully reduced the complexity to be polynomial time by employing realistic subnetwork assumptions. Efficient algorithms for maximizing probabilities of success in the presence of deadlines are also demonstrated.

5 Acknowledgements

This work was partially supported by Air Force Office of Scientific Research grants F49620-97-1-0382 and F49620-95-1-0305. The authors thank D. Bertsekas and S. Patek for pointing out some related work and references.

References

[AIS] J. Ambros-Ingerson and S. Steel. Integrating planning, execution and monitoring *In Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88),* Morgan Kaufmann, St. Paul. Minnesota, 1972

[B57] Richard .E. Bellman. *Dynamic programming,* Princeton University Press, Princeton, New Jersey 1957

[B87] D. M. Bertsekas. *Dynamic Programming,* Prentice Hall, Englewood Cliffs , NJ, 1987.

[B85] D. M. Bertsekas. *Dynamic Programming and Optimal Control,* Athena Scientific, Belmont, Massachusetts, 1995.

[CT] K.W. Currie and A. Tate. O-Plan: The Open Planning Architecture *Artificial Intelligence,* 52(1). 1991

[CGM] G. Cybenko, R. Gray, and K. Moizumi. Q-learning: A Tutorial and Extensions *Mathematics of Neural Networks,* Kluwer Academic Publishers, Boston/London/Dordrecht, 1997

[DB] M. Drummond and J. Bresian. Anytime synthetic projection: Maximizing the probability of goal satisfaction *In Proceedings of AAAI-90,* Boston, 1990

[FN] R. E. Fikes and N. J. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving *Artificial Intelligence,* 2(3-4), 1971

[GJ] M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, San Francisco, 1979

[G] Robert Gray. Agent Tcl : A flexible and secure mobile-agent system *Proceeding of the 1996 Tcl/Tk Workshop,* July 1996.

[H] R. A. Howard. *Dynamic programming and Markov Processes,* MIT Press, Cambridge, Massachusetts, 1960

[J] J. R. Jackson. "Scheduling a production line to minimize maximum tardiness" *Research Report 43, Management Science Research Project* University of California, Los Angeles, 1955

[JvRS] Dag Johansen, Robbert van Renesse, and Fred B. Scheidner. Operating system support for mobile agents *In Proceeding of the 5th IEEE Workshop on Hot Topics in Operating Systems,* 1995

[K] R. M. Karp. Reducibility among combinatorial problems in R.E. Miller and J.W. Thatcher (eds.), *Complexity of Computer Computations,* Plenum Press, New York, 1972

[RRK] D. Rus, R. Gray, and D. Kotz. Autonomous and adaptive agents that gather information *In AAAI 96 International Workshop on Intelligent Adaptive Agents,* August 1996.

[S] E. D. Sacerdoti. The nonlinear nature of plans *In Proceedings of the Fourth International Joint Conference on Artificial Intelligence(IJCAL75),* Tbilisi, Georgia, 1975

[W] J. E. White. Telescript technology: The foundation for the electronic marketplace *General Magic White Paper,* General Magic, 1994

[P] S. Patek. Ingress planning in FASM, ALPHATECH Technical Report, Burlington, MA, 1997.