Chapter 29
The Canadian Traveller Problem

Amotz Bar-Noy* Baruch Schieber*

Abstract

Suppose that a road map is given in which each road is associated with the time it takes to traverse it. However, this road map is unreliable; some of the roads might be unsuitable for travel at certain times, and such blockage would be revealed only upon reaching an adjacent site. The Canadian Traveller Problem is to devise a travel strategy that would guarantee a good path between two sites given this uncertainty. Papadimitriou and Yannakakis proved that if the number of roads that might be blocked is not fixed, then devising a strategy that guarantees a given competitive ratio is PSPACE-complete. In this paper, we study several variations of this problem.

- In the Recoverable Canadian Traveller Problem, each site is associated with a recovery time to reopen any blocked road that is adjacent to it. We present a polynomial-time travel strategy that guarantees the shortest worst-case travel time, in the case where an upper bound on the number of blockages is known in advance, and the recovery times are not long relative to the travel times.
- In the stochastic variation of the Recoverable Canadian Traveller Problem, each road is associated with an independent probability of being blocked. For this problem, we present a polynomial-time strategy that minimizes the expected travel time, in the case where the recovery times are not long relative to the travel times.
- Another variation considered is the k-Canadian Traveller Problem. Here, an upper bound, k, on the number of blocked roads is given as a parameter. We present a travel strategy that guarantees the shortest worst-case travel time. The time complexity of devising this strategy is polynomial for any constant k. On the other hand, we prove that devising such a strategy when k is non-constant is PSPACE-complete.
- A "dual" problem to the Canadian Traveller Problem is the k-Vital Edges Problem. Given a road map and two specified sites, find k roads whose blockage would maximize the increase in the travel time between the two sites. We prove that this problem is NP-hard, even when the travel time of all the roads is constant.

The Canadian Traveller Problem and its variations can be viewed as routing problems. In many communication networks messages routing is done according to an outdated topology of the network. Our algorithms could replace existing routing schemes to achieve more robust routing.

*IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY 10598.

1. introduction

This paper considers several problems related to the Canadian Traveller Problem (CTP). The CTP has been introduced in [PY89] and is defined as follows: Suppose that a traveller has to go from site s to site t. The traveller knows a graph (a map) $G(V, E)$, where the set of nodes V corresponds to the set of sites, and the set of edges E corresponds to the set of roads between sites. In addition, each edge $e \in E$ is associated with

a non-negative real length $\ell(e)$ interpreted as the time it takes to traverse the road e. However, this map is unreliable; some of the roads may be unsuitable for travel at certain times (e.g., blocked by snowfall), and such blockage of a road would be revealed only upon reaching an adjacent site. The problem is to devise a *travel strategy* that results in an optimal travel (according to some predefined measure) from s to t given this uncertainty.

The main difficulty in devising a good travel strategy stems from the on-line nature of the problem; decisions have to be made based on partial information with no knowledge of future blockages. Notice that devising an off-line travel strategy is simple: when all road blockages are known in advance, the optimal travel strategy is given by the shortest path from s to t.

The design of algorithms for on-line problems received a lot of attention recently (see, e.g., [ST85, MMS88, BCR87, PY89]). The quality of an on-line algorithm is usually measured based on two alternate criteria. The first criterion is the *competitive ratio* of the algorithm [ST85]: the ratio of the performance of the on-line algorithm to the performance of the off-line algorithm for the same instance. The second criterion is the traditional worst-case performance [BB90]. The difference between these two criteria in our problem is demonstrated by the following example. Suppose that the traveller has to go from Victoria to Quebec. Assume that in one possible scenario the off-line strategy results in ten days travel, while in another possible scenario the off-line strategy results in two days travel. Following the competitive ratio criterion, the traveller would prefer an on-line strategy that guarantees twenty days travel in the first scenario as long as this strategy guarantees four days travel in the second scenario. Following the worst-case criterion, the traveller would prefer a strategy that always results in fifteen days travel. It seems that each one of these criteria may be useful under some circumstances.

For the CTP, Papadimitriou and Yannakakis [PY89] showed that devising an on-line algorithm with a bounded competitive ratio is PSPACE-complete. The problem remains hard even when a blockage probability for each road is given in advance.

In this paper we consider the worst-case criterion. We study several variations of the CTP. The first variation is the Recoverable Canadian Traveller Problem (Recoverable-CTP). In the original problem, once a road is blocked, it remains blocked forever. In the Recoverable-CTP, blocked roads may be reopened. Specifically, each node $x \in V$ is associated with a non-negative real time $\ell(x, x)$ interpreted as the *recovery time* of edges adjacent to x. Whenever the traveller, at a node x, finds a blocked edge e, he/she can either use another edge or wait $\ell(x, x)$ time and then use e (provided that e is not found to be blocked again). Note that the CTP is a special case of the Recoverable-CTP, where all the recovery times are very large.

We consider deterministic and stochastic variations of the Recoverable-CTP. In the deterministic variation, we assume that the number of road blockages that might occur during a travel is bounded. Observe that if we drop this assumption, there is no point in waiting for reopening of a road, since as soon as a road is reopened it may become blocked again. In the stochastic variation each edge $e \in E$ is associated with a probability $0 \leq p(e) < 1$, interpreted as the blockage probability of road e. Each time the traveller reaches one of the adjacent sites of road e, the probability of finding this road blocked is $p(e)$.

Next, we study the k-Canadian Traveller Problem (k-CTP). In this problem, as in the original problem, blocked roads cannot be reopened. The difference is that in the k-CTP, the maximum number of road blockages that may occur is given as a parameter. Notice that the CTP is a special case of the k-CTP, when this parameter is taken to be the number of edges. The rationale for considering the k-CTP is that usually this parameter may assumed to be small relative to the number of edges. We note that it is common to parameterize the number of failures in fault tolerant problems (e.g., the Byzantine Generals Problem [PSL80]).

A problem that is "dual", in some sense, to the k-CTP is the k-Vital Edges Problem (k-VEP) defined as follows (see [CS82, BGV89, MMG89]). Given a weighted graph $G(V, E)$ with two specified nodes s and t, and a parameter k, find k edges whose removal from the graph would maximize the increase in the shortest path between s and t. The blockage of these edges result in the worst-case scenario for the off-line k-CTP defined above. It was not known whether the k-VEP is in P. In [MMG89] a polynomial-time algorithm is described for the 1-VEP. The exponential time algorithm for the k-VEP that is described in the same paper seems to be false.

We present the following results:

The stochastic Recoverable-CTP: let n and m be the number of nodes and edges in G. We present an $O(m \log n)$ time algorithm for designing a travel strategy from all sites to a fixed destination that guarantees the shortest expected travel time, under the assumption that $\ell(x, x) \leq \ell(e)$, for any e adjacent to x.

The deterministic Recoverable-CTP: let k be the bound on the number of road blockages. We give an $O((k^2 m + kn \log n))$ time algorithm for designing a travel strategy from all sites to a fixed destination that guarantees the shortest worst-case travel time, under the assumption that $\ell(x, x) \leq \ell(e)$, for any e adjacent to x.

The k-CTP: let k be the bound on the number of blockages that may occur during a travel. We prove that for arbitrary k the problem of designing a travel strategy that guarantees the shortest travel time is PSPACE-complete. For $k = 1$, we give an $O(m + n \log n)$ time algorithm for designing this strategy. This algorithm can be extended to give a polynomial-time travel strategy for any constant k.

The k-VEP: we prove that the corresponding decision problem is *strong* NP-complete; that is, NP-complete even for *unit* edge lengths.

We note that all our results can be extended to directed graphs, and to the case where nodes can be blocked too.

The Canadian Traveller Problem and its variations can be viewed as an abstraction for some routing problems. In many communication networks, the routing of messages is done along the shortest paths according to local knowledge of the topology of the network. (See, e.g. the routing scheme of ARPANET [MRR80, Ros80].) The topology of the network is updated periodically. If the topology does not change between periodical updates, then such a routing scheme is optimal. Each message is sent along the shortest path from its source to its destination. However, things may become worse if even one link fails. Solving the Recoverable-CTP and the k-CTP result in more robust routing schemes for such networks. For example, applying our solution to k-CTP would result in a routing algorithm that achieves the shortest worst-case arrival time of a message in case up to k links are disconnected between two periodical topology updates. (On the other hand, nothing is guaranteed when more than k links are disconnected.) The solution to the Recoverable-CTP can be used in networks where the time of recovering a crashed node in the network is known.

2. The stochastic Recoverable-CTP

In this section, we present a travel strategy for the stochastic Recoverable-CTP. In this problem, each edge, $e \in E$, is associated with a length $\ell(e)$ and a blockage probability $p(e)$. Each time the traveller at x tries to traverse the edge e, the probability that e is not blocked is $1 - p(e)$, where all these probabilities are independent. If e is blocked, the traveller may either try to proceed along some other edge or wait $\ell(x, x)$ time for the recovery of e. Our solution is based on the assumption that $\ell(x, x) < \ell(x, y)$, for all neighbors y of x. We leave open the question whether this assumption could be relaxed.

Fix a destination node t. Without loss of generality, we can view the operations taken by the traveller at x in any travel strategy as composed of several phases. In each phase the traveller uses a

priority list $x_1, x_2, \ldots, x_j$ of x's neighbors. When the phase starts, the traveller tries to traverse the edge (x, x_1), if (x, x_1) is blocked, the traveller tries to traverse the edge (x, x_2), and so forth. If all the trials fail, the traveller remains at x and the next phase starts. Notice that given all the priority lists and the expected travel time from x_i to t, for the x_is in all the lists, the expected travel time from x to t can be computed.

Next, we prove some properties of optimal travel strategies. To break symmetry, the strategies we consider obey the following rules: (i) For a node x, if remaining at x or traveling to some neighbor of x result in the same expected travel time, then the traveller would prefer to remain at x. (ii) An arbitrary order is imposed on the neighbors of x. If y precedes z in this order, and traveling from x to either y or z results in the same expected travel time, then the traveller would prefer to travel to y.

The first lemma states that if a traveller going from x to t arrives at y with some positive probability, then a traveller going from y to t cannot arrive at x. To formalize this we need the following definition.

Definition 1: *Let R be an optimal travel strategy. Define the directed graph $G_R(V, E_R)$ as follows. For each undirected edge $(x, y) \in E$, the directed edge $(x \rightarrow y)$ is in E_R if y is in some priority list of x.*

Lemma 2.1: *The directed graph G_R is acyclic (dag).*

Proof: (sketch) Let $E(x)$ denote the expected travel time from x to t. Assume to the contrary that there exists a cycle in G_R, and let $(x \rightarrow y)$ be an edge on such a cycle. We claim that $E(x) > E(y)$. To see this, suppose that $E(x) \leq E(y)$. Define $0 < p_{xy}$ to be the probability that a traveller going from x to t arrives at y. Hence, for some L,

$$E(x) = p_{xy}(\ell(x, y) + E(y)) + (1 - p_{xy})L.$$

Consider the travel strategy in which the traveller remains at x instead of traveling to y (in the first time it arrives x). Then, the expected travel time is

$$E'(x) = p_{xy}(\ell(x, x) + E(x)) + (1 - p_{xy})L.$$

Since $\ell(x, x) < \ell(x, y)$ and $E(x) \leq E(y)$ it follows that $E(x) > E'(x)$. This contradicts the optimality of R. We conclude that $E(x) > E(y)$. Since the claim holds for any edge on the directed cycle $y_0 \rightarrow y_1 \rightarrow \cdots \rightarrow y_m \rightarrow y_0$ it follows that $E(y_0) > E(y_1) > \cdots > E(y_k) > E(y_0)$; a contradiction. □

Let $x_1, \ldots, x_k$ be the sequence of the neighbors of x for which $E(x_i) < E(x)$, sorted in increasing ordered according to $\ell(x, x_i) + E(x_i)$ (ties are broken by the predefined order). Next, we prove that the priority lists are prefix of this sequence.

Lemma 2.2: *Any priority list of x in R is a prefix of the above sorted sequence.*

Proof: (sketch) Suppose that x_i appears before x_j in the priority list. We claim that $i < j$. Otherwise, the expected travel time can be decreased by exchanging x_i with x_j in the list. Now, consider the i-th node in the priority list of x. We claim that this node is x_i. Assume to the contrary that i is the minimal index such that the i-th node in the list is x_j for some $j > i$. Then, either omitting x_j from the list, or adding x_i before x_j decreases the expected travel time; a contradiction to the optimality of the strategy. □

The proof of the following lemma will be presented in the full version.

Lemma 2.3: *There is an optimal travel strategy in which for each node all its priority lists are identical.*

Following the above lemmata, the priority list of x, and consequently $E(x)$ is computed as follows. Suppose that $E(x_i)$ is known for all the k neighbors of x for which $E(x_i) < E(x)$. Let $p_i = p(x, x_i)$, $q_i = 1 - p_i$. Let $P_0 = 1$, and

$P_i = \prod_{j=1}^{i} p_j$, for $i = 1, \ldots, k$. Define $\alpha_i = \sum_{j=1}^{i} P_{j-1} q_j (\ell(x, x_j) + E(x_j))$. Given that the priority list of node x consists of the first h edges, the expected travel time is

$$\frac{\alpha_h + P_h \ell(x, x)}{1 - P_h}.$$

Thus, we choose h to be the index minimizing this value. It can be shown that this is the minimum index h for which

$$\frac{\alpha_h + \ell(x, x)}{1 - P_h} < \ell(x, x_{h+1}) + E(x_{h+1}).$$

Also, it can be shown that if

$$\frac{\alpha_h + \ell(x, x)}{1 - P_h} < \ell(x, x_{h+1}) + E(x_{h+1}),$$

then

$$\frac{\alpha_{h+1} + \ell(x, x)}{1 - P_{h+1}} < \ell(x, x_{h+2}) + E(x_{h+2}).$$

Consequently, h can be found by a binary search.

The computation of the priority lists is done by a labeling algorithm analogue to Dijkstra's algorithm for finding shortest paths [Dij59]. In the algorithm, a set L of labeled nodes is maintained. For each $x \in L$, its priority list and $E(x)$ are known. Initially, $L = \emptyset$. For each node $x \in V - L$, there is a tentative priority list which is the optimal list among all lists consisting only of nodes in L. Given this tentative list, $tentative - E(x)$, the expected travel time to t given the tentative priority list, is computed. Initially, $tentative - E(t) = 0$ and $tentative - E(x)$ for each $x \in V - \{t\}$ is set to infinity. Each iteration of the algorithm consists of two steps: a labeling step and an updating step.

Labeling step: Find the node x with the minimum $tentative - E(x)$, and add it to L making its tentative values final.

Updating step: Let $Adj(x)$ be the set of nodes adjacent to x. For each $y \in Adj(x) - L$ check if inserting the edge (y, x) to its priority list improves its tentative expected travel time. If so, update its values accordingly. Specifically, insert the node x to the sorted list of the neighbors of y, according to the value of $\ell(y, x) + E(x)$. Then, find the prefix of this list that gives the best expected travel time.

The algorithm terminates when all the nodes are labeled; that is when $L = V$.

We omit the proofs of the following correctness and optimality lemmata.

Lemma 2.4: *For any node x, the expected travel time from x to t in the above travel strategy is $E(x)$ as computed by the algorithm.*

Lemma 2.5: *In any travel strategy and for any node x, the expected travel time from x to t is greater or equal to $E(x)$.*

Complexity: The labeling algorithm can be implemented using Fibonacci heaps [FT87]. However, the updating step takes $O(\log n)$ per edge. We get that the complexity of the implementation is $O(m \log n)$ time for a fixed destination t.

3. The deterministic Recoverable-CTP

Suppose that we are given a graph $G(V, E)$ with a non-negative length $\ell(e)$ associated with each edge $e \in E$ and a non-negative recovery time $\ell(x, x)$ for each node $x \in V$, where $\ell(x, x) \leq \ell(e)$, for any edge adjacent to x. In this section, we describe a travel strategy that solves the deterministic Recoverable-CTP, where at most k edges might be blocked, for some fixed parameter k. Our strategy has the best performance among all possible on-line travel strategies and can be computed efficiently.

Fix some destination t. Consider the case $k = 0$, i.e., no edge can be blocked. Then, when the traveller arrives at a node x, the best performance is achieved by proceeding along the first edge in a shortest path from x to t.

Assume now that $k > 0$. We compute the travel strategy recursively. That is, to compute the k-th travel strategy we assume that all the

strategies for $0 \leq i < k$ have been already computed. Let $dist(i,x)$, $0 \leq i \leq k$, be the worst-case travel time from x to t given that at most i edges may be blocked. Suppose that the traveller aiming to t arrives at x. We distinguish between two cases.

CASE 1: h blocked edges have been discovered before arriving x, for some $1 \leq h \leq k$. Our assumption that the recovery time is shorter than the time it takes to traverse any edge adjacent to the endpoint of a blocked edge guarantees that on the arrival to x all the discovered blocked edges are recovered. Because of this, and since only $k-h$ additional edges can be blocked, the strategy for $k-h$ (that has been computed in the recursion) can be used.

CASE 2: No edges have been blocked so far. In this case, the traveller checks how many edges adjacent to x are blocked. If no edge is found to be blocked then he/she traverses some edge $(x, x_{0,0}^{(k)})$. We call this edge the *primary edge* of x.

Suppose that i edges are found to be blocked, for some $1 \leq i \leq k$, then the traveller has two possibilities: (i) The traveller may wait at x for the recovery of its adjacent edges causing an addition of $\ell(x,x)$ to the travel time. After the waiting the travel strategy for $k-i$ is used. (ii) The traveller may traverse some unblocked edge.

The strategy is given by a list of $h+1 \leq i+1$ edges, called *alternate edges*, and denoted $(x, x_{i,j}^{(k)})$, for $j = 0, \ldots, h$. If the list consists of less than $i+1$ edges, then the last one must be (x,x). (To make the length of each list $i+1$, we define $(x, x_{h+1}) = \cdots = (x, x_i) = (x,x)$, if $h < i$.) When the traveller finds i blocked edges at x, he/she selects the unblocked edge $(x, x_{i,j}^{(k)})$ with the minimal index j. If this edge is (x,x) then the traveller waits at x, otherwise it traverses the selected edge. Notice that at least one edge in this list is not blocked.

First, we show how to compute the list of the alternate edges $(x, x_{i,0}^{(k)}), \ldots, (x, x_{i,i}^{(k)})$, for any $0 < i \leq k$. Given these lists the primary edges are computed. If (x,y) is taken as the first alternate edge, then the worst-case travel time from x to t if this edge is used is $\ell(x,y) + dist(k-i,y)$. Thus, $x_{i,0}^{(k)}$ is taken to be the node minimizing $\{\ell(x,y) + dist(k-i,y)\}$ over $y \in \{x\} \cup Adj(x)$. If $x_{i,0}^{(k)} = x$ then the traveller remains at x. Suppose that $x_{i,a}^{(k)} \neq x$ for all $0 \leq a < j \leq i$. The j-th alternate edge $(x, x_{i,j}^{(k)})$ is given by the node $x_{i,j}^{(k)}$ minimizing $\{\ell(x,y) + dist(k-i,y)\}$ over $y \in \{x\} \cup Adj(x) - \cup_{a=0}^{j-1} \{x_{i,a}^{(k)}\}$. Notice that out of the $i+1$ edges that has to be computed for each $0 \leq i \leq k$, i have been already computed in the recursion. To see this we state the following lemma.

Lemma 3.1: *The edge $(x, x_{i,j}^{(k)})$ is the same as the edge $(x, x_{i-1,j}^{(k-1)})$, for $0 \leq j \leq i-1$.*

Now, we turn to describe how to compute the primary edges. The worst-case travel time is given either by not blocking any primary edges and thus letting the traveller traverse the *primary route*, or by blocking some edges of any intermediate node x, causing the traveller to proceed along one of the alternate edges of x or to remain at x. Using this observation, it can be shown that $dist(k,x)$, i.e., the worst-case travel time from x is given by $\max_{i=0}^{k} \{\ell(x, x_{i,i}^{(k)}) + dist(k-i, x_{i,i}^{(k)})\}$. (Notice that the worst-case for i blocked edges is given when all the edges $(x, x_{i,j}^{(k)})$, for $0 \leq j \leq i-1$, are blocked.) The primary edge is the edge that minimizes this maximum (ties are broken by some predefined order).

We claim that the primary edges form a tree similar to the shortest path tree. The values $primary(x)$ and $dist(k,x)$, for all nodes $x \in V$ are found by a labeling algorithm analogue to Dijkstra's algorithm for finding shortest paths [Dij59].

In the algorithm, a set L of labeled nodes is maintained; initially, $L = \emptyset$. For each $x \in L$, $dist(k,x)$ and $primary(x)$ are known. For each $x \in V - L$, $tentative - dist(k,x)$ and $tentative - primary(x)$ are the tentative values for these variables. Initially, $tentative - dist(k,t) = 0$ and $tentative - dist(k,x)$ is set to infinity, for each

$x \in V - \{t\}$. Each iteration of the algorithm consists of two steps: a labeling step and an updating step.

Labeling step: Find the node x with the minimum *tentative* $- dist(k, x)$, and add it to L making its tentative values final. (Note that t is the first to join L.)

Updating step: For each $y \in Adj(x) - L$, check if making the edge (y, x) y's primary edge improves the tentative travel time from y. If so, update y's values accordingly. Specifically, for each such vertex y, we check the worst-case travel time resulting from using the primary edge (y, x). This time is

$$\max \quad \{ \quad \ell(y,x) + dist(k,x), \quad \max_{i=1}^{h} \{\ell(y, y_{i,i}^{(k)}) + dist(k-i, y_{i,i}^{(k)})\} \quad \},$$

where $(y, y_{1,1}^{(k)}), \ldots, (y, y_{h,h}^{(k)})$ are the alternate edges of y.

The algorithm terminates when all the nodes are labeled; that is when $L = V$.

The proofs of the following correctness and optimality lemmata are omitted.

Lemma 3.2: *If primary(x) is taken to be the primary edge of x, for each $x \in V$, then, $dist(k, x)$ computed by the algorithm is the worst-case travel time from x to t given that at most k edges may be blocked.*

Lemma 3.3: *For any assignment of primary edges and for each $x \in V$, the worst-case travel time given that at most k edges may be blocked is greater or equal to $dist(k, x)$.*

Complexity: The algorithm for k, given the output for $0 \leq i < k$, can be implemented using Fibonacci heaps similar to Dijkstra's algorithm [FT87] in $O(km + n \log n)$ time, for a fixed destination t. This gives an overall time complexity of $O(k^2 m + kn \log n)$.

4. The k-Canadian Traveller Problem

Suppose that we are given a graph $G(V, E)$ with a non-negative length $\ell(e)$ associated with each edge $e \in E$. In addition, we are given a parameter k that bounds the number of edge blockages that may occur during a travel. The k-CTP is to devise a travel strategy that guarantees the shortest worst-case travel time between two nodes s to t. The main result of this section is the proof that the problem of devising such a strategy for an arbitrary k is PSPACE-complete. We also present an efficient algorithm for devising a travel strategy between all the nodes and some node t, for $k = 1$. Our strategy has the best performance among all possible on-line travel strategies. The running time of this algorithm is the same as the running time of the algorithm for computing the shortest paths tree of t. Finally, we show how this algorithm can be extended to an (exponential) algorithm for arbitrary k.

Theorem 4.1: *The k-Canadian Traveller Problem is PSPACE-complete.*

Proof: (outline) The proof is by reducing the Quantified Satisfiability problem (QSAT) to the k-CTP. As in [PY89], the problem is viewed as a game between a searcher and an adversary. We show that the searcher can find a strategy that guarantees a given travel time if and only if the input quantified formula is satisfiable. The reduction is similar to the reduction that is described in [PY89]. However, some non-trivially modifications are necessary. One of the reasons for this is that in our problem the adversary has a bound on the number of edges it may block. Thus, we have to make sure that if the formula is not satisfiable, the adversary will still be able to block all the short paths from s to t without blocking more than k edges. □

Now, we briefly describe the algorithm for devising a strategy for the k-CTP when $k = 1$. The strategy is based on the following simple observation. Suppose that the traveller is currently at node x and she/he finds one of the edges (x, y) to

be blocked. Then, since no more edges may be blocked, the traveller may take the shortest path from x to t in $G(V, E - \{x, y\})$. This observation implies a way to compute the *alternate edges*: the edges along which the traveller goes after an edge has been already found to be blocked. It remains to show how to compute the *primary edges*: the edges along which the traveller goes when no edge has been yet found to be blocked.

Definition 2: *For $F \subseteq E$, let $\bar{G}(F)$ be the subgraph induced by the subset $E - F$. Define shortest(F, x) to be the length of the shortest path from x to t in $\bar{G}(F)$.*

Suppose that

$$P = (s = x_0, x_1), (x_1, x_2), \ldots, (x_a, x_{a+1} = t)$$

is the path from s to t composed of primary edges. This would be the path taken by the traveller if no edge has been found to be blocked. Then, the worst-case travel time is the maximum between $\sum_{i=0}^{a} \ell(x_i, x_{i+1})$ and

$$\max_{0<j\leq a} \{\sum_{i=1}^{j} \ell(x_{i-1}, x_i) + shortest(\{(x_j, x_{j+1})\}, t)\}.$$

The first term corresponds to the scenario where no edge has been found to be blocked, while the rest correspond to the scenarios where some edge has been found to be blocked.

To find the primary edges that achieve the shortest worst-case travel time, a labeling algorithm is employed. For this, we compute $shortest(\{(x, y)\}, x)$ for all $x \in V$ and $(x, y) \in E$; that is the shortest paths from all nodes x to t in the graph resulting by blocking any edge adjacent to x. All these computations can be done in $O(m + n \log n)$ time using Dijkstra's algorithm as implemented in [FT87], and some additional Fibonacci heaps.

To get the algorithm for arbitrary k we need the following definition.

Definition 3: *For $F \subseteq E$, $|F| \leq k$, let $dist(F, x)$ be the shortest worst-case travel time from x to t that can be achieved when all the edges in F are already found to be blocked and the total number of blocked edges is at most k.*

As in the previous section the strategy at each node can be represented as a priority list of k edges, the first of which is the *primary edge*. Suppose that

$$P = (s = x_0, x_1), (x_1, x_2), \ldots, (x_a, x_{a+1} = t)$$

is the path from s to t composed of primary edges. Then, the worst-case travel time is the maximum between $\sum_{i=0}^{a} \ell(x_i, x_{i+1})$ and

$$\max_{0<j\leq a} \{\sum_{i=1}^{j} \ell(x_{i-1}, x_i) + dist(\{(x_j, x_{j+1})\}, x_j)\}.$$

Thus, the computation of the primary edges requires the computation of $dist(\{(x, y)\}, x)$, for each edge $(x, y) \in E$. This requires, recursively, the computation of $dist(\{e, (x, y)\}, x)$, for all edges $e \in E$. Overall, it requires the computation of $dist(F, x)$ for all $x \in V$ and all subsets $F \subseteq E$ of size at most k, containing an edge adjacent to x, for all nodes x.

5. The k-Vital Edges Problem

The k-Vital Edges Problem (k-VEP) is defined as follows. Given a weighted graph $G(V, E)$ with two specified nodes s and t, and a parameter k. Find k edges whose removal from G maximizes the increase in the shortest path between s and t. In this section, we prove that the k-VEP problem is NP-hard.

Consider the decision problem corresponding to the k-VEP problem. In the decision problem an additional bound b is given. The goal is to decide whether there are k edges whose removal from the graph increases the shortest path between s to t to be longer than b.

Theorem 5.1: *The decision k-Vital Edges Problem is* strong *NP-complete.*

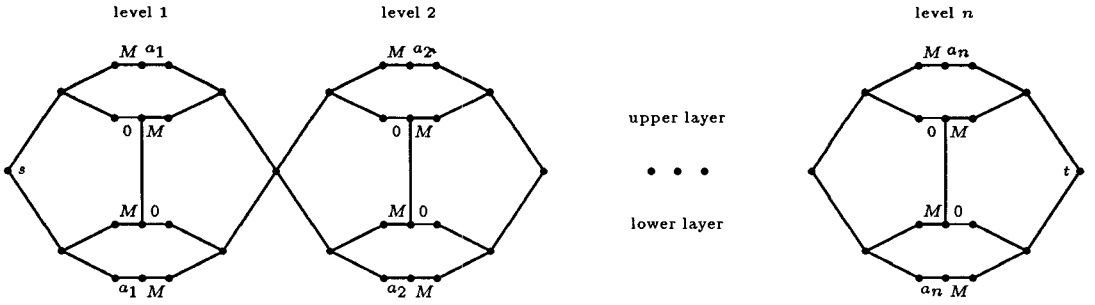

Figure 1: The graph G in the reduction from the Partition problem.

The proof is by reducing the Three Dimensional Matching Problem (3DM) to the decision k-VEP where the input graph has unit weights. The reduction is quite complicated and thus omitted. Instead, a simpler theorem stating that this problem is (weak) NP-complete is proved. Some of the tools used in this proof are used in the proof of Theorem 5.1.

Theorem 5.2: *The decision k-Vital Edges Problem is NP-complete.*

Proof: (sketch) The proof is by reducing the Partition problem to the k-VEP problem. In the Partition problem, the inputs are n positive numbers $a_1, a_2, \ldots, a_n$, and the goal is to decide whether there exists a subset S of indices such that $\sum_{i \in S} a_i = \sum_{j \notin S} a_j$.

Given an instance of the Partition problem, we construct a corresponding instance of the decision k-VEP such that the answer for the Partition instance is positive if and only if the answer for the corresponding k-VEP instance is positive. The parameter k for this k-VEP instance is n and the bound b is $(n + \frac{1}{2}) \sum_{i=1}^{n} a_i$.

Figure 1 depicts the weighted graph G for this instance. The label close to an edge denotes its weight where all unlabelled edges have weight zero. All the edges except those who are labelled with zero cannot be disconnected. This is achieved by multiplying these edges $n+1$ times. The value of M is $\sum_{i=1}^{n} a_i$.

Suppose that n edges are removed from G. The following lemmata are given without a proof.

Denote the edge labelled with zero in the upper layer of level i by e_i^{up}. Similarly, denote this edge in the lower layer by e_i^{down}.

Lemma 5.3: *If there exists a level i in which both e_i^{up} and e_i^{down} are not removed, then there exists a path from s to t whose length is strictly smaller than nM.*

Lemma 5.4: *If there exists a level i in which both e_i^{up} and e_i^{down} are removed, then there exists a level j in which both e_j^{up} and e_j^{down} are not removed.*

Lemma 5.5: *If for each level i, either e_i^{up} or e_i^{down} are removed, then the length of the shortest path from s to t is at least nM.*

Corollary 5.6: *In order for the shortest path to be at least nM, exactly one edge out of e_i^{up} and e_i^{down} has to be removed, for each level i.*

Suppose that one edge out of e_i^{up} and e_i^{down} is removed, for each level i.

Lemma 5.7: *The shortest path from s to t is contained in either the upper layer or the lower layer.*

Let P^{up} be the shortest path from s to t that is contained in the upper layer. Similarly, let P^{down} be the path that is contained in the lower layer. Let UP be the set of all levels i in which the edge e_i^{up} is removed.

Lemma 5.8: *The length of P^{up} is $nM + \sum_{i \in UP} a_i$ and the length of P^{down} is $nM + \sum_{i \notin UP} a_i$.*

Corollary 5.9: *The length of the shortest path from s to t is $nM + \min\{\sum_{i\in UP} a_i, \sum_{i\notin UP} a_i\}$.*

We conclude that the length of the shortest path from s to t is at most $b = nM + \frac{1}{2}\sum_{i=1}^{n} a_i$. The length of this path is b if and only if there exists a set UP such that $\sum_{i\in UP} a_i = \frac{1}{2}\sum_{i=1}^{n} a_i$; that is, iff the Partition problem has a positive answer. □

Acknowledgement. We thank Michael Ben-Or, Shay Kutten, Prabhakar Raghavan, and Marc Snir for helpful discussions.

References

[BB90] S. Ben-David and A. Borodin. A new measure for the study of online algorithms, 1990. Manuscript.

[BCR87] R.A. Baeza-Yates, J.C. Culberson, and G.J.E. Rawlins. Searching with uncertainty. Technical Report CS-87-68, Computer Science Department, The University of Waterloo, October 1987.

[BGV89] M.O. Ball, B.L. Golden, and R.V. Vohra. Finding most vital arcs in a network. *Operations Research Letters*, 8:73–76, April 1989.

[CS82] H.W. Corley and D.Y Sha. Most vital links and nodes in weighted networks. *Operations Research Letters*, 1:157–160, September 1982.

[Dij59] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1:269–271, 1959.

[FT87] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34:596–615, July 1987.

[MMG89] K. Malik, A.K. Mittal, and S.K. Gupta. Finding most vital arcs in a network. *Operations Research Letters*, 8:223–227, April 1989.

[MMS88] M.S. Manasse, L.A. McGeoch, and D.D. Sleator. Competitive algorithms for on-line problems. In *Proc. of the 20th ACM Symp. on Theory of Computing*, pages 322–333, Chicago, IL, May 1988.

[MRR80] J.M. McQuillan, I. Richer, and E.C. Rosen. The new routing algorithm for the ARPANET. *IEEE Transactions on Communications*, COM-28:711–719, May 1980.

[PSL80] M. Pease, R.E. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.

[PY89] C.H. Papadimitriou and M. Yannakakis. Shortest paths without a map. In *Proc. 16th ICALP, Lect. Notes in Comp. Sci.* No. 372, pages 610–620. Springer-Verlag, July 1989.

[Ros80] E.C. Rosen. The updating protocol of ARPANET's new routing algorithm. *Computer Networks*, 4:11–19, February 1980.

[ST85] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.