Canadians Should Travel Randomly*

Erik D. Demaine[1], Yamming Huang[2], Chung-Shou Liao[2], and Kunihiko Sadakane[3]

[1] Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139, USA
edemaine@mit.edu

[2] Department of Industrial Engineering and Engineering Management, National Tsing Hua University, Hsinchu 30013, Taiwan
s100034528@m100.nthu.edu.tw, csliao@ie.nthu.edu.tw

[3] National Institute of Informatics, Tokyo, Japan
sada@nii.ac.jp

Abstract. We study online algorithms for the Canadian Traveller Problem (CTP) introduced by Papadimitriou and Yannakakis in 1991. In this problem, a traveller knows the entire road network in advance, and wishes to travel as quickly as possible from a source vertex s to a destination vertex t, but discovers online that some roads are blocked (e.g., by snow) once reaching them. It is PSPACE-complete to achieve a bounded competitive ratio for this problem. Furthermore, if at most k roads can be blocked, then the optimal competitive ratio for a deterministic online algorithm is $2k+1$, while the only randomized result known is a lower bound of $k+1$.

In this paper, we show for the first time that a polynomial time randomized algorithm can beat the best deterministic algorithms, surpassing the $2k+1$ lower bound by an $o(1)$ factor. Moreover, we prove the randomized algorithm achieving a competitive ratio of $\left(1+\frac{\sqrt{2}}{2}\right)k+1$ in pseudo-polynomial time. The proposed techniques can also be applied to implicitly represent multiple near-shortest s-t paths.

1 Introduction

Imagine attempting to drive across your favorite northern country in the dead of winter. Snow is falling in unpredictable patterns, effectively blocking certain roads from passage (either from lack of plowing or accident pile-ups). You have purchased a complete road map, modeled as a weighted graph $G = (V, E)$ whose edges represent roads and whose edge weights represent the time to traverse the edge. But you have no knowledge of which roads are blocked by weather or accidents, until you reach a vertex incident to such a road, in which case you can directly observe the blockage before attempting traversal. This problem, called the Canadian Traveller Problem (CTP), was defined by Papadimitriou

* Supported by NSC Grant 102-2221-E-007-075-MY3 and KAKENHI 23240002

and Yannakakis [14] in 1991. The objective is to design an efficient route from a source to a destination under this condition of uncertainty. The major difficulty in developing a good strategy based on partial information is to make decisions without being able to predict blockages.

Prior work on CTP. Papadimitriou and Yannakakis [14] proved that it is PSPACE-complete to devise a CTP strategy that guarantees a bounded competitive ratio. They also proved that the stochastic model of this problem, in which a probability that each edge is blocked, independent of all others, is given in advance, is #P-hard when minimizing the expected competitive ratio to the offline optimum [14]. Bar-Noy and Schieber [1] investigated several variations of the CTP from the worst-case perspective, where the objective is to find a static (offline) algorithm that minimizes the maximum travel cost [2]. They considered the k-CTP in which the number of blockages is bounded by k. Note that for an arbitrary k, the problem of designing a strategy that guarantees a given travel time remains PSPACE-complete, as has been shown in [1, 14].

During recent decades, no significant progress has been made in the development of online approximation algorithms for solving the k-CTP. Basically, two simple deterministic strategies for solving this problem are available [16, 17]. The *greedy* algorithm (GA) starts at a vertex v and finds the shortest v-t path using Dijkstra's algorithm [5] in a greedy manner based on the current blockage information. The other strategy, called the *reposition* algorithm (RA), proposed by Westphal [16], requires the traveller to begin at the source s, follow the shortest s-t path until she learns about a blockage on the path to t, and then returns to s and takes a new shortest s-t path based on the updated blockage information. Westphal [16] proved that no deterministic online algorithm within a $(2k + 1)$-competitive ratio exists for the problem, and the simple reposition algorithm achieves the lower bound. Westphal also proved a lower bound of $k+1$ on the competitive ratios of all randomized online algorithms. Xu et al. [17] developed a similar deterministic adaptive *comparison* strategy that incorporates the concept of reposition; the approach achieves the tight deterministic lower bound as well. They also showed that the competitive ratio of the GA algorithm is exponential in k in the worst case. Huang and Liao [9] considered a generalization of the k-CTP, called the Double-valued Graph, in which each edge is associated with two possible distances. They proposed lower bounds and a simple algorithm that meets the deterministic lower bound. They also extended the k-CTP to the design of a tour through a set of vertices, in which the traveller visits each vertex and returns to the origin under the same uncertainty.

Our results. We develop improved randomized strategies for solving the k-CTP. Whereas deterministic algorithms have been extensively studied, research on randomized approaches for solving the problem is lacking.[1] In this paper, we propose a polynomial time randomized algorithm that surpasses the deterministic lower bound of $2k + 1$ by an $o(1)$ factor. In addition, the competitive ratio of

[1] Recently, Bender and Westphal proposed a randomized algorithm using the RA strategy for special graphs in which all s-t paths are vertex-disjoint [3].

this algorithm can be improved to $\left(1+\frac{\sqrt{2}}{2}\right)k+1$ in pseudo-polynomial running time. This result is the first demonstration that randomization strictly helps in the k-CTP for arbitrary graphs.

The rationale behind the proposed randomized algorithm is summarized as follows. Given a connected graph $G=(V,E)$ with a source s and a destination t in V and a distance function $d: E \to R^+$, the algorithm first selects a set S of near-shortest s-t paths whose distance cost does not exceed the product of that of the shortest s-t path and a threshold factor. Precisely, this set comprises all s-t paths of cost $(1+\alpha)d(s,t)$, where $d(s,t)$ denotes the shortest travel cost from s to t and α is a small constant. Let the set of all such paths be represented by an *apex tree* T, which is a tree-like graph that becomes a tree by removal of a vertex. Then, the traveller traverses T rather than the original graph G using an online randomized strategy under the same uncertainty until all possible s-t paths in S are blocked. We repeat the argument until she arrives at t.

Near-shortest paths. In order to conduct the randomized algorithm in polynomial time, we need to find all near-shortest s-t paths efficiently, which is of independent interest. There has been a considerable amount of research investigating the problem. Eppstein's well-known approach for finding ℓ shortest s-t paths or all s-t paths shorter than a given distance cost, in a directed graph $G=(V,A)$, spends constant time for each of the ℓ paths, after a fast preprocessing step that runs in $O(|V|\log|V|+|A|)$ time [6]. That is, the ℓ shortest paths can be obtained in $O(\ell)$ time. Note that ℓ may be exponential in $|V|$, even if all the ℓ paths have the same distance. In Eppstein's study, cycles of repeated vertices were allowed. Recently, Hershberger et al. [8] and Frieder and Roditty [7] studied finding the ℓ shortest simple (i.e., loopless) paths in directed graphs. In undirected graphs, Katoh et al. [11] investigated finding ℓ shortest simple paths in an undirected graph $G=(V,E)$, and their algorithm, which takes $O(\ell(|V|\log|V|+|E|))$ time, is the best known-to-date result.

In this paper, we propose an implicit representation, constructed in $O(\mu^2|E|^2)$ time and $O(\mu|E|)$ space, of all *strictly* jth-shortest s-t paths, $1 \le j \le \mu$, where μ is at most the summation of distances of all edges. That is, assume $d^1(s,t), d^2(s,t), d^3(s,t), \ldots$ denotes the strictly increasing sequence of all possible distinct s-t path distances, where $d^1(s,t)=d(s,t)$; our technique can represent all strictly jth-shortest paths of cost $d^j(s,t)$, $1 \le j \le \mu$. Note that a strictly jth-shortest s-t path can be obtained by finding ℓ shortest s-t paths for a sufficiently large value of ℓ; however, the worst-case number of such near-shortest s-t paths can be exponential in the order of G. The proposed implicit representation can guarantee the pseudo-polynomial running time of the randomized strategy.

2 Preliminaries

Given a connected graph $G=(V,E)$ with a source s and a destination t, let an s-t path p of length m be $p: s=v_1-v_2-\cdots-v_m-v_{m+1}=t$. We denote the subset of blockages in E identified by an online algorithm A during the trip as

$E_i^A = \{e_1, e_2, \ldots, e_i\} \subseteq E$, $1 \leq i \leq k$, where e_i is the ith blockage identified, and let $E_0 = \emptyset$ and E_k be the set of all blocked edges. Let $d_{E_i^A}(s,t)$ denote the travel cost from s to t, derived by an adaptive algorithm A that learns about blockage information E_i during the trip; and let $d_{E_k}(s,t)$ be the offline optimum from s to t under complete information E_k. For convenience, we denote by $d_{E_i}(s,t)$-*path* a route along which the traveller spends at most $d_{E_i}(s,t)$. For all instances, the following property is immediately obtained, where $E_1 \subseteq E_2 \subseteq \cdots \subseteq E_k$.

$$d(s,t) = d_{E_0}(s,t) \leq d_{E_1}(s,t) \leq \cdots \leq d_{E_k}(s,t). \tag{1}$$

We refer to [4, 15] and formally define the competitive ratio as follows. An online randomized algorithm A is c^A-*competitive* against an oblivious adversary for the k-CTP if

$$\mathrm{E}[d_{E_i^A}(s,t)] \leq c^A \cdot d_{E_k}(s,t) + \varepsilon, \quad 1 \leq i \leq k,$$

where $\mathrm{E}[d_{E_i^A}(s,t)]$ is the expected travel cost of the randomized strategy A and c^A and ε are constants. To analyze the performance of online algorithms for the k-CTP, we make two basic assumptions [1, 17]: one is that once a blocked edge is discovered by the traveller, the edge remains blocked forever. The other is that the given graph G remains connected even if all the blockages are eliminated.

3 Main Algorithm

Given a connected graph $G = (V, E)$ with a source s and a destination t, we require a set S of all $(1+\alpha)d_{E_i}(s,t)$-paths from s to t under blockage information E_i, where α is a constant, $0 < \alpha < 1$. In contrast to the previous studies, we find strictly jth-shortest paths, $1 \leq j \leq \mu$, for a sufficiently large μ, to derive the set S of the s-t paths. The technique for so doing will be presented later.

Algorithm 1: Greedy & Reposition Randomized Algorithm (GRR)

Input : A graph $G = (V, E)$ with a source s and a sink t, and constants k and α;
Output : A random route from s to t;

Let $i = 0$; ▷ no blockage found
while the traveller does not arrive at t **do**
    Find a set S of all $(1+\alpha)d_{E_i}(s,t)$-paths from s to t;
    Randomly select an s-t path from S to traverse with the following probabilities;
    $\frac{k-i}{k-i+1}$: she leaves for t along any $d_{E_i}(s,t)$-path until a blockage is found;
    $\frac{1}{k-i+1}$: she follows **Traverse-Tree** on the remaining s-t paths in S;
    if the traveller learns about the jth blockage on the way to t, $j < k$ **then**
        the traveller returns to s and $i \leftarrow j$;
    else if the traveller learns about the last blockage **then** ▷ finding all blockages
        the traveller returns to s and follows a $d_{E_k}(s,t)$-path to t;
    end if
end while

The main algorithm (Algorithm 1) is described as follows. First, we select a set S of all $(1+\alpha)d_{E_i}(s,t)$-paths from s to t under blockage information E_i, $0 \le i \le k$, for a small α. Then, the traveller uses the *reposition* RA strategy, and when restarting at s, she either selects an arbitrary $d_{E_i}(s,t)$-path from S and traverses the path with probability $\frac{k-i}{k-i+1}$, or she chooses the remaining s-t paths in S and traverses them using the Traverse-Tree procedure with probability $\frac{1}{k-i+1}$. The algorithm repeats until the traveller arrives at t.

To analyze the competitive ratio of the entire GRR algorithm, the traveller is supposed to be able to efficiently traverse the remaining s-t paths in S following the Traverse-Tree procedure, whose details will be introduced in the next section. More precisely, if the extra travel cost of the procedure is bounded within an acceptable range for every blockage discovered, then the ratio can be improved over the deterministic lower bound of $2k+1$. The correctness of the claim will be proved in the next section.

Claim 1: If the traveller uses the Traverse-Tree procedure on the $(1+\alpha)d_{E_i}(s,t)$-paths from s to t in S and arrives at t, $0 \le i < k$, each blockage increases the total travel cost of the algorithm by at most $(1+\alpha)d_{E_k}(s,t)$ on average.

Theorem 1. *The k-Canadian Traveller Problem can be approximated within a competitive ratio $\left(1+\frac{\sqrt{2}}{2}\right)k+1$, when the number of blockages is up to a given constant k.*

Proof. In each iteration, r, of the while loop, let the cost of conducting the Traverse-Tree procedure be $c(r)$ for blockages discovered. When the traveller arrives at t using the GRR algorithm, consider the case whether she learns about all the blockages or not.

Case 1: The traveller does not learn about every blockage when she arrives at t; i.e. at least one $(1+\alpha)d_{E_{k-1}}(s,t)$-path from s to t is unblocked during the trip.

In the worst case analysis, assume the traveller learns about only one blockage in each iteration of the while loop. Based on Claim 1, $c(r) \le (k-r+1)(1+\alpha)d_{E_k}(s,t)$ for every iteration r of the loop, where the number of remaining blockages undiscovered is $k-r+1$ in iteration r. Consequently, the expected travel cost of the GRR algorithm is formulated as follows:

$$
\begin{aligned}
\mathrm{E}[d_{E_k^{GRR}}(s,t)] \le{}& \left[\frac{k}{k+1}\cdot 2d(s,t)+\frac{1}{k+1}\cdot c(1)\right]+\frac{k}{k+1}\left[\frac{k-1}{k}\cdot 2d_{E_1}(s,t)+\frac{1}{k}\cdot c(2)\right]\\
&+\frac{k}{k+1}\cdot\frac{k-1}{k}\cdot\left[\frac{k-2}{k-1}\cdot 2d_{E_2}(s,t)+\frac{1}{k-1}\cdot c(3)\right]+\cdots\cdots\\
&+\left[\frac{1}{k+1}\cdot 2d_{E_{k-1}}(s,t)+\frac{1}{k+1}\cdot c(k)\right]+d_{E_k}(s,t)\\
\le{}&\left(\frac{k}{k+1}+\frac{k-1}{k+1}+\cdots+\frac{1}{k+1}\right)\cdot 2d_{E_{k-1}}(s,t)\\
&+\left(\frac{k}{k+1}+\frac{k-1}{k+1}+\cdots+\frac{1}{k+1}\right)\cdot(1+\alpha)d_{E_k}(s,t)+d_{E_k}(s,t)\\
\le{}&\left[k+\frac{1}{2}(1+\alpha)k+1\right]\cdot d_{E_k}(s,t).
\end{aligned}
$$

Thus, in Case 1, the competitive ratio of the GRR algorithm is at most

$$\frac{\left[k + \frac{1}{2}(1+\alpha) \cdot k + 1\right] \cdot d_{E_k}(s,t)}{d_{E_k}(s,t)} = \frac{3+\alpha}{2} \cdot k + 1.$$

Case 2: The traveller learns about all the k blockages before she arrives at t; that is, each $(1+\alpha)d_{E_{k-1}}(s,t)$-path is blocked during the trip. Thus, she eventually restarts at s and follows a $d_{E_k}(s,t)$-path to t, as indicated in Line 8 of the GRR algorithm.

The condition implies that the distance of an offline optimal s-t path is $d_{E_k}(s,t) > (1+\alpha)d_{E_{k-1}}(s,t)$. Moreover, Claim 1 cannot be applied because the traveller cannot reach t while conducting the Traverse-Tree procedure. Thus, an upper bound on the travel cost $c(r)$ in iteration r is derived by exploiting the RA strategy [16]. In the worst case analysis, $c(r) \leq 2(k-r+1)(1+\alpha)d_{E_{k-1}}(s,t)$ for every iteration r, where the number of remaining blockages undiscovered is $k-r+1$ in iteration r. The expected total travel cost of the GRR algorithm is formulated in a similar manner:

$$\mathrm{E}[d_{E_k^{GRR}}(s,t)] \leq k(2+\alpha) \cdot d_{E_{k-1}}(s,t) + d_{E_k}(s,t).$$

Hence, in Case 2, the competitive ratio of the GRR algorithm is at most

$$\frac{k(2+\alpha) \cdot d_{E_{k-1}}(s,t) + d_{E_k}(s,t)}{d_{E_k}(s,t)} \leq \frac{k(2+\alpha) \cdot d_{E_{k-1}}(s,t)}{(1+\alpha) \cdot d_{E_{k-1}}(s,t)} + 1 = \frac{2+\alpha}{1+\alpha} \cdot k + 1.$$

By simple algebra, the competitive ratio of the GRR algorithm can be minimized in the two cases by letting the constant α be $\sqrt{2}-1$. So the expected competitive ratio is at most $\left(1+\frac{\sqrt{2}}{2}\right)k+1$. Note that for any small constant α', $0 < \alpha' \leq \alpha = \sqrt{2}-1$, the competitive ratio is still strictly smaller than the deterministic lower bound of $2k+1$. □

4 Apex Tree

In this section, we consider the Traverse-Tree procedure conducted in a tree-like graph, called an *apex tree*, which can represent all $(1+\alpha)d_{E_i}(s,t)$-paths from s to t, $0 \leq i < k$, in a given graph G, for a small constant α.

A graph $T = (V, E)$ is an *apex tree* if T comprises a source vertex s, a rooted tree that consists of a destination vertex t (as root) and all other vertices in V, and edges that connect s to each leaf and some internal vertices of the tree. That is, $T \setminus \{s\}$ is actually a tree that is rooted at t.

Subsequently, we claim that the Traverse-Tree procedure (see Algorithm 2) is an optimal randomized strategy for solving the k-CTP in an apex tree T, if the distance cost of every s-t path in T is assumed to be identical. Note that the worst-case instance that was reported in [16] to establish the lower bound of $k+1$ is in fact also an apex tree where all s-t paths have the same cost. Thus,

Algorithm 2: TRAVERSE-TREE

Input : An apex tree T that represents all $(1+\alpha)d_{E_i}(s,t)$-paths from s to t;
Output : A random route from s to t;

Assign equal probabilities to every child of the root t, and sequentially repeat the process for each descendant of t in the order of breadth-first search;
The traveller begins at s, and randomly selects an s-t path according to the assigned probability and leaves for t following that path;
while the traveller does not arrive at t **do**
    Let the blocked edge discovered by the traveller be $e = (v_i, v_{i+1})$ along a path $p : s = v_1 - v_2 - \cdots - v_h = t$;
    The traveller returns to s and eliminates the blocked s-v_i path;
    while every s-t path through the vertex v_{i+1} is currently blocked **do**
        $i \leftarrow i + 1$; ▷ depth-first search order of the path p
    end while
    Reassign probabilities only to the subtree rooted at v_{i+1} in a similar way;
    The traveller randomly selects an s-t path through v_{i+1} according to the assigned probability and leaves for t following that path;
end while

the competitive ratio of the algorithm can achieve the lower bound and it follows that the algorithm is optimal.

The main concept of the Traverse-Tree procedure is to incorporate randomized operations into the reposition RA strategy and then to explore subtrees of an apex tree T in the order of *depth-first search*. Precisely, we initially distribute probabilities of path selection equally among every child of the root t. We sequentially distribute the probabilities equally to each descendant in the order of *breadth-first search*. When the traveller starts at the source s, she randomly selects an s-t path according to the assigned probability and follows the path to t. While finding a blockage on the way to t, the traveller uses the RA strategy and returns to s. We eliminate the blocked path, and reassign probabilities to the unblocked subtrees similarly. The traveller traverses the remaining routes in T by exploring the subtrees in the order of *depth-first search*. The argument repeats until the traveller arrives at the destination t.

For the k-CTP in an apex tree T, there is at least one s-t path without a blockage. Let this offline optimal s-t path be $p : s = v_1^* - v_2^* - \cdots - v_m^* = t$ and the number of children of a vertex v_j^* be c_j in the apex tree T, such that v_j^* has children $v_{j,1}, v_{j,2}, \ldots, v_{j,c_j}$. Assume the last child of each v_j^*, v_{j,c_j}, $2 \le j \le m$, lies on the path p; i.e. $v_{j,c_j} = v_{j-1}^*$. Moreover, suppose each subtree rooted at $v_{j,\ell}$, $1 \le \ell \le c_j - 1$, has $b_{j,\ell}$ blockages. Consider the expected total cost when the traveller traverses the paths other than the offline optimal path. Note that the malicious adversary does not block any edge $(s, v) \in E$. So $\sum_{j=3}^{m} \sum_{\ell=1}^{c_j-1} b_{j,\ell} = k$.

Lemma 1. *For the k-CTP in an apex tree T in which the distance costs of all s-t paths are equal, there is an optimal $(k+1)$-competitive randomized algorithm.*

Proof. Let $\mathrm{E}(s, v_i^*)$ be the expected total travel cost from s to v_i^*. For $t = v_m^*$, we evaluate the cost. If $c_m = 1$, we obtain $\mathrm{E}(s, v_m^*) \le \mathrm{E}(s, v_{m-1}^*) + d_{E_k}(v_{m-1}^*, v_m^*)$.

If $c_m > 1$, with probability $\frac{1}{c_m}$, the traveller finds v^*_{m-1} as a predecessor of v^*_m in the first trial. Then the expected travel cost is at most $\mathrm{E}(s, v^*_{m-1}) + d_{E_k}(v^*_{m-1}, v^*_m)$. If the traveller cannot find v^*_{m-1} in the first trial, with probability $\left(1 - \frac{1}{c_m}\right)\frac{1}{c_m - 1}$, she finds v^*_{m-1} in the second trial. Without loss of generality, suppose the traveller selects $v_{m,\ell}$ from $\ell = 1$ to $\ell = c_m - 1$, when finding $v^*_{m-1} = v_{m,c_m}$. In this case the expected travel cost is $\left\{2b_{m,1}d_{E_{b_{m,1}}}(s, v_{m,1}) + \mathrm{E}(s, v^*_{m-1}) + d_{E_k}(v^*_{m-1}, v^*_m)\right\}$ because the reposition algorithm may return to s $b_{m,1}$ times and find the way to v^*_{m-1} with $\mathrm{E}(s, v^*_{m-1})$ expected cost and finally go to v^*_m with cost $d_{E_k}(v^*_{m-1}, v^*_m)$. Therefore we obtain

$$
\begin{aligned}
\mathrm{E}(s, v^*_m) \le\ & \tfrac{1}{c_m}\left\{\mathrm{E}(s, v^*_{m-1}) + d_{E_k}(v^*_{m-1}, v^*_m)\right\} \\
& + (1 - \tfrac{1}{c_m})\tfrac{1}{c_m - 1}\left\{2b_{m,1}d_{E_{b_{m,1}}}(s, v_{m,1}) + \mathrm{E}(s, v^*_{m-1}) + d_{E_k}(v^*_{m-1}, v^*_m)\right\} \\
& + (1 - \tfrac{1}{c_m})(1 - \tfrac{1}{c_m - 1})\tfrac{1}{c_m - 2}\left\{2b_{m,1}d_{E_{b_{m,1}}}(s, v_{m,1}) + 2b_{m,2}d_{E_{b_{m,2}}}(s, v_{m,2})\right. \\
& \left. + \mathrm{E}(s, v^*_{m-1}) + d_{E_k}(v^*_{m-1}, v^*_m)\right\} + \cdots \\
& \le \tfrac{1}{c_m}\left\{2(b_{m,1} + b_{m,2} + \cdots + b_{m,c_m-1})d_{E_k}(s, t)\right\} + \mathrm{E}(s, v^*_{m-1}) \\
& + d_{E_k}(v^*_{m-1}, v^*_m) \\
& \le \sum_{\ell=1}^{c_m - 1} b_{m,\ell}d_{E_k}(s, t) + \mathrm{E}(s, v^*_{m-1}) + d_{E_k}(v^*_{m-1}, v^*_m)
\end{aligned}
$$

Therefore either $c_m = 1$ or not, we obtain $\mathrm{E}(s, v^*_m) \le \sum_{\ell=1}^{c_m - 1} b_{m,\ell}d_{E_k}(s, t) + \mathrm{E}(s, v^*_{m-1}) + d_{E_k}(v^*_{m-1}, v^*_m)$. Similarly, $\mathrm{E}(s, v^*_{m-1}) \le \sum_{\ell=1}^{c_{m-1} - 1} b_{m-1,\ell}d_{E_k}(s, t) + \mathrm{E}(s, v^*_{m-2}) + d_{E_k}(v^*_{m-2}, v^*_{m-1})$. Therefore, $\mathrm{E}(s, v^*_m) \le \sum_{j=2}^{m}\sum_{\ell=1}^{c_j - 1} b_{j,\ell}d_{E_k}(s, t) + d_{E_k}(v^*_1, v^*_2) + \ldots + d_{E_k}(v^*_{m-1}, v^*_m) = (k+1)d_{E_k}(s, t)$, because $\sum_{\ell=1}^{c_2 - 1} b_{2,\ell} = 0$. The expected total cost of the algorithm, including the distance cost of the offline optimal path, is $(k+1)d_{E_k}(s, t)$. The competitive ratio achieves the lower bound for any randomized online algorithms in apex trees. □

Based on the above proof, if the distance cost of each s-t path in T is at most $(1+\alpha)d_{E_i}(s, t)$, $0 \le i < k$, then the upper bound on the expected total travel cost of the algorithm is $\sum_{j=3}^{m}\sum_{\ell=1}^{c_j - 1} b_{j,\ell}(1+\alpha)d_{E_k}(s, t) = k(1+\alpha)d_{E_k}(s, t)$, excluding the cost of the offline optimal path. The next theorem follows immediately and proves the claim made in Section 3, i.e., each blockage increases the total travel cost by at most $(1 + \alpha)d_{E_k}(s, t)$ on average.

Theorem 2. *For the k-CTP in an apex tree T in which each s-t path is a $(1 + \alpha)d_{E_i}(s, t)$-path, $0 \le i < k$, the competitive ratio of the Traverse-Tree procedure is at most $(1 + \alpha)k + 1$.*

5 Implicit Representation of Near-shortest Paths

Given a connected undirected graph $G = (V, E)$ with a source s and a destination t, we give two simple data structures for storing all shortest to strictly μth-shortest s-t paths, for a large value of μ. The first representation is for storing

shortest to strictly jth-shortest simple s-t paths, provided that $d^j(s,t)$ is given, $1 \leq j \leq \mu$, and the second one is for representing possibly non-simple paths whose distances are from the cost $d(s,t)$ to $d^\mu(s,t)$.

5.1 Strictly second-shortest paths

For each vertex $v \neq s$, let $S^j(v)$ be the vertex set that comprises all v's predecessors, each of which is v's preceding neighbor lying in a strictly ith-shortest s-v path, $1 \leq i \leq j$. To represent all shortest to strictly jth-shortest s-t paths, we define the jth-*shortest path digraph*, denoted by $D^j(G) = (V^j, A^j)$ of G, where an arc $\overrightarrow{(u,v)} \in A^j$ if and only if there exists a strictly ith-shortest s-t path p in G, $1 \leq i \leq j$, such that $(u,v) \in p$; all isolated vertices are eliminated in V^j. The previous studies [10, 12, 13] investigated the strictly second-shortest path problem (i.e., *next-to-shortest path*) based on the *shortest path digraph*, i.e., $D^1(G) = (V^1, A^1)$. Notably, $D^1(G)$ is acyclic and can be constructed in $O(|V|^2)$ time [10, 12]. Moreover, for every vertex $v \neq s$, $d^1(s,v)$ and $S^1(v)$ can also be obtained. Kao et al.'s algorithm [10] can derive the cost of a strictly second-shortest s-t path, $d^2(s,t)$ in $O(|V|^2)$ time, given the shortest path digraph $D^1(G)$.

1: **procedure Find-2nd-Shortest**$(G, s, t, D^1(G))$ ▷ find each $d^2(s,v)$ and $S^2(v)$
2: Use Kao et al.'s algorithm to compute $d^2(s,t)$ based on $D^1(G)$;
3: Initialize a queue $Q = \{t\}$, $D^2(G) = (V^1, \emptyset)$, and $S^2(v) = \emptyset$, $\forall v \in V$;
4: **while** the queue $Q \neq \emptyset$ **do**
5: $u \leftarrow$ Dequeue(Q);
6: **for** each neighbor w adjacent to u and $\overrightarrow{(u,w)} \notin A^2$ **do** ▷ breath-first search
7: **if** $d^2(s,u) - d(w,u) \geq d^1(s,w)$ **then**
8: $A^2 \leftarrow A^2 \cup \overrightarrow{(w,u)}$ and $S^2(u) \leftarrow S^2(u) \cup \{w\}$;
9: **if** $d^2(s,u) - d(w,u) > d^1(s,w)$ **or**
$d^2(s,u) - d(w,u) = d^1(s,w)$ and $w \in V \setminus V^1$ **then**
10: $d^2(s,w) \leftarrow d^2(s,u) - d(w,u)$ and $V^2 \leftarrow V^2 \cup \{w\}$;
11: **end if**
12: Enqueue(Q, w) if w is not in the queue Q;
13: **end if**
14: **end for**
15: **end while**
16: **end procedure**

We present the Find-2nd-Shortest procedure to search for all strictly second-shortest simple s-t paths and to construct the representation $D^2(G) = (V^2, A^2)$. The main step of this procedure is just a breadth-first search. We start at t and traverse backward all other vertices until s, and to determine whether each vertex lies on a strictly second-shortest s-t path. The correctness of the procedure follows from the optimal substructure property; moreover, the resulting graph $D^2(G)$, which is a directed acyclic graph from s to t, can represent an apex tree.

Note that in an apex tree, there is a unique path from each vertex to t, while in $D^2(G)$ there may exist multiple paths to t. However, this is not an obstacle to use of our algorithm in $D^2(G)$ if we obtain a random path from t to s. The traveller randomly selects one of incoming edges (or outgoing edges) to a vertex, and traverses the edge. The traveller excludes already visited vertices when she randomly selects a path.

In addition, the procedure can be generalized to finding all strictly third-shortest to μth-shortest simple s-t paths, provided that the cost $d^j(s,t)$ is given, $3 \leq j \leq \mu$; note that the property holds between every strictly $(j-1)$th-shortest and jth-shortest paths. However, Kao et al.'s method cannot be straightforwardly extended to compute $d^j(s,t)$, $j \geq 3$. We leave it as an open problem to find an efficient way to derive $d^j(s,t)$, $j \geq 3$.

Clearly, the data structure $D^2(G)$ can be constructed in polynomial time and linear space. In each iteration of the loop in the GRR algorithm, we find all shortest to strictly second-shortest s-t paths whose cost is assumed to be at most $d^2(s,t) = (1+\alpha')d(s,t)$, for some $\alpha' > 0$. The competitive ratio would be $(\frac{2+\alpha'}{1+\alpha'})k + 1 < 2k + 1$, and therefore, the GRR algorithm can surpass the deterministic lower bound for the k-CTP in polynomial time.

1: **procedure Find-Multiple-Shortest**(G, s, t, μ)
2: Duplicate each edge in the input graph $G = (V, E)$ to make G directed;
3: Let $S_0 = \{t\}$, $D_V(t) = \{0\}$; $D_E(\overrightarrow{(u,v)}) = \{\infty\}$ $\overrightarrow{(u,v)} \in E$; $D_V(v) = \{\infty\}$ $v \in V$;
4: **for** $i = 0$ to $\mu|E|$ **do**
5: Let $S_{i+1} = \emptyset$;
6: **for** each $v \in S_i$ **do**
7: **for** each $e = \overrightarrow{(u,v)} \in E$ **do**
8: $S_{i+1} \leftarrow S_{i+1} \cup \{u\}$;
9: Let $L = \{d(u,v) + \ell \mid \ell \in D_V(v)\}$;
10: Let $D_E(\overrightarrow{(u,v)})$ be the set of the smallest μ values in $D_E(\overrightarrow{(u,v)}) \cup L$;
11: **end for**
12: **end for**
13: **for** each $u \in V$ **do**
14: **for** each $e = \overrightarrow{(u,v)} \in E$ **do**
15: Let $D_V(u)$ be the set of the smallest μ values in $D_V(u) \cup D_E(\overrightarrow{(u,v)})$;
16: **end for**
17: **end for**
18: **end for**
19: **end procedure**

5.2 Strictly μth-shortest paths

We compute the sets of possibly non-simple strictly jth-shortest paths for $j = 1, 2, \ldots, \mu$ using the Find-Multiple-Shortest procedure. This is again a breadth-first search traversing from t backward to s. We have some notation. In the ith

iteration, we keep a set S_i of vertices which are reached from t using exactly i edges, and initially, $S_0 = \{t\}$. Let a set $D_V(u)$ store shortest to strictly μth-shortest distances from u to t; that is, $D_V(s) = \{d^1(s,t), d^2(s,t), \ldots, d^\mu(s,t)\}$. Let two sets $D_E(\overrightarrow{(u,v)})$ and $D_E(\overrightarrow{(v,u)})$ for an edge $e = (u,v)$ store distances from u to t and distances from v to t, respectively, using the edge e. Precisely, $\ell \in D_E(\overrightarrow{(u,v)})$ for an edge $e = (u,v)$ if and only if there is a u-t path of distance $\ell \in D_V(u)$ through the edge e. Note that the breadth-first search traverses a vertex multiple times. It is enough to repeat $\mu|E|$ times the iteration because a strictly μth-shortest s-t path uses at most $\mu|E|$ edges.

---

procedure Implicit-Representation(G, s, t, L)
  Let $i = 0$, $S_i = \{s\}$, $V' = \{s\}$, $E' = \emptyset$, $L(v) = \emptyset$ $\forall v \in V$, and $L(s) = D_V(s) \cap L$;
  while $S_i \neq \emptyset$ **do**
    Let $S_{i+1} = \emptyset$;
    for each $u \in S_i$ **do**
      for each $e = \overrightarrow{(u,v)} \in E$ **do**
        Let $L = \{\ell - d(u,v) \mid \ell \in L(u)\}$;
        $L(v) \leftarrow L(v) \cup (D_V(v) \cap L)$;
        if $L(v) \neq \emptyset$ **then**
          $S_{i+1} \leftarrow S_{i+1} \cup \{v\}$, $V' \leftarrow V' \cup \{v\}$, and $E' \leftarrow E' \cup \{e\}$;
        end if
      end for
    end for
    $i \leftarrow i + 1$;
  end while
end procedure

Based on this data structure, we can construct a graph $G' = (V', E')$ which represents all (possibly non-simple) s-t paths whose distances are in a given set $L \subseteq \{d^1(s,t), \ldots, d^\mu(s,t)\}$ using the Implicit-Representation procedure. Starting from s, we traverse an edge $e = (u,v)$ only if the set $D_E(\overrightarrow{(u,v)})$ or $D_E(\overrightarrow{(v,u)})$ of distances to t contains the given set of distances. Similarly, in the ith iteration, we keep a set S_i of vertices which are reached from s using exactly i edges, and initially, $S_0 = \{s\}$. Let $L(v)$ be the set of distances from v to t for each vertex $v \in V'$. Precisely, if $\ell \in L(v)$, then $\ell \in D_V(v)$ and there is a v-t path of cost ℓ. The number of iterations of the procedure is also at most $\mu|E|$.

The resulting graph G' is a directed graph from s to t. However because there may exist non-simple paths, the graph may have cycles. We can convert G' into a directed acyclic graph G'' as follows. For each $\ell \in L(v)$, we create a vertex v_ℓ, and for each pair of vertices v_ℓ and $w_{\ell'}$, we create an edge from v_ℓ to $w_{\ell'}$ if and only if $(v,w) \in E'$ and $\ell - d(v,w) = \ell'$. The graph G' is converted so that vertices can be duplicated to eliminate multiple edges. Each vertex is identified with the name of the vertex in the original graph and the unique distance to t.

Then the new graph containing those newly inserted vertices and edges becomes a directed acyclic graph. Therefore, the graph G'' can represent an apex tree.

In summary, the simple implicit representation has at most $\mu|V|$ vertices and at most $\mu|E|$ edges, and it can be constructed in $O(\mu^2|E|^2)$ time. Hence, we can obtain a set of paths whose distances are at most $(1+\alpha)d(s,t)$ by setting μ to be the summation of distances of all edges. The number of iterations of the loop in the *GRR* algorithm is at most k and thus the whole process takes $O(k\mu^2|E|^2)$ time in the worst case; that is, the proposed $\left(1+\frac{\sqrt{2}}{2}\right)k+1$-competitive randomized algorithm runs in pseudo-polynomial time.

References

1. A. Bar-Noy and B. Schieber. The Canadian traveller problem. In *Proc. of the 2nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, (1991), pp. 261–270.
2. S. Ben-David and A. Borodin. A new measure for the study of online algorithms. *Algorithmica*, (1994), 11(1), pp. 73–91.
3. M. Bender and S. Westphal. An optimal randomized online algorithm for the k-Canadian traveller problem on node-disjoint paths. *Journal of Comb. Opt.*, (2013), Published online.
4. A. Borodin and R. El-Yaniv. Online computation and competitive analysis. Cambridge University Press, Cambridge, (1998).
5. E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, (1959), 1(1) pp. 269–271.
6. D. Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, (1998), 28(2), pp. 652–673.
7. A. Frieder and L. Roditty. An experimental study on approximating k shortest simple paths. *Proc. of 19th Annual European Symp. on Algorithm (ESA)*, (2011), LNCS 6942, pp. 433–444.
8. J. Hershberger, M. Maxel and S. Suri. Finding the k shortest simple paths: A new algorithm and its implementation. *ACM Trans. on Algorithms*, (2007), 3(4), p.45.
9. Y. Huang and C.S. Liao. The Canadian traveller problem revisited. In *Proc. of 23th International Symposium on Algorithms and Computation (ISAAC)*, (2012), LNCS 7676, pp.352–361.
10. K.H. Kao, J.M. Chang, Y.L. Wang and J.S.T. Juan. A quadratic algorithm for finding next-to-shortest paths in graphs. *Algorithmica*, (2011), 61, pp. 402-418.
11. N. Katoh, T. Ibaraki, and H. Mine. An efficient algorithm for k shortest simple paths. *Networks*, (1982), 12(4), pp. 411–427.
12. I. Krasikov and S.D. Noble. Finding next-to-shortest paths in a graph. *Information Processing Letters*, (2004), 92, pp. 117–119.
13. K.N. Lalgudi and M.C. Papaefthymiou. Computing strictly-second shortest paths. *Information Processing Letters*, (1997), 63, pp. 177–181.
14. C.H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, (1991), 84(1), pp. 127–150.
15. D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, (1985), 28, pp. 202–208.
16. S. Westphal. A note on the k-Canadian traveller problem. *Information Processing Letters*, (2008), 106, pp. 87–89.
17. Y.F. Xu, M.L. Hu, B. Su, B.H. Zhu, and Z.J. Zhu. The Canadian traveller problem and its competitive analysis. *Journal of Comb. Opt.*, (2009), 18, pp.195–205.