Approximating the Canadian Traveller Problem with Online Randomization

Algorithmica manuscript No.
(will be inserted by the editor)

Approximating the Canadian Traveller Problem with Online Randomization

Erik D. Demaine · Yamming Huang ·
Chung-Shou Liao* · Kunihiko Sadakane

Received: date / Accepted: date

Abstract In this paper, we study online algorithms for the Canadian Traveller Problem (CTP) defined by Papadimitriou and Yannakakis in 1991. This problem involves a traveller who 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. Achieving a bounded competitive ratio for the problem is PSPACE-complete. Furthermore, if at most k roads can be blocked, the optimal competitive ratio for a deterministic online algorithm is $2k + 1$, while the only randomized result known so far is a lower bound of $k + 1$.

We show, for the first time, that a polynomial time randomized algorithm can outperform the best deterministic algorithms when there are at least two blockages, and surpass the lower bound of $2k + 1$ by an $o(1)$ factor. Moreover, we prove that the randomized algorithm can achieve a competitive ratio of $\left(1 + \frac{\sqrt{2}}{2}\right)k + \sqrt{2}$ in pseudo-polynomial time. The proposed techniques can also be exploited to implicitly represent multiple near-shortest s-t paths.

An extended abstract of this paper appeared in the proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP 2014).
This work was partially supported by MOST Taiwan under Grants MOST105-2628-E-007-010-MY3, MOST105-2221-E-007-085-MY3, and JSPS KAKENHI 23240002.

Erik D. Demaine
Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139, USA
E-mail: edemaine@mit.edu

Yamming Huang · Chung-Shou Liao*
Department of Industrial Engineering and Engineering Management, National Tsing Hua University, Hsinchu 30013, Taiwan
E-mail: s100034528@m100.nthu.edu.tw; *csliao@ie.nthu.edu.tw

Kunihiko Sadakane
Department of Mathematical Informatics, the University of Tokyo, Tokyo, Japan
E-mail: sada@mist.i.u-tokyo.ac.jp

Keywords Canadian traveller problem · competitive ratio · randomized algorithm · next-to-shortest path · strictly ith-shortest path

1 Introduction

Imagine a person attempting to drive across a country in the Northern Hemisphere in the dead of winter. Snow is falling in unpredictable patterns, and some roads are blocked due to the lack of snow plows or accident pile-ups. The driver has purchased a complete road map, modeled as an edge-weighted graph $G = (V, E)$ in which the edges represent the roads and each edge weight represents the time required to traverse that edge. However, the driver has no knowledge of roads that are blocked due to the weather or accidents until he/she reaches a vertex incident to such a road. Then, he/she can observe the blockage directly before attempting traversal. The problem is called the Canadian Traveller Problem (CTP), and was defined by Papadimitriou and Yannakakis [22]. The objective is to design an efficient route from a source to a destination under conditions of uncertainty. The major difficulty in developing a good strategy based on partial information is the need to make decisions without being able to predict blockages.

Previous work on CTP. The CTP is actually a two-player game between a traveller and a malicious adversary who sets up road blockages to maximize the ratio between the performance of the online strategy and that of the offline optimum in which the blocked edges are eliminated from the graph. Papadimitriou and Yannakakis [22] proved that devising a CTP strategy that guarantees a bounded competitive ratio is PSPACE-complete. They also proved that the stochastic model of the problem (in which the probability that each edge is blocked, independent of all other edges, is given in advance) is #P-hard when minimizing the expected competitive ratio to the offline optimum. 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 shown in [1,22]. In addition, Bar-Noy and Schieber discussed the Recoverable k-CTP in which each blocked edge is associated with a recovery time (which is not very long relative to the traversal time) for reopening. Subsequently, Karger and Nikolova [18] studied the stochastic CTP in special graph classes and developed exact algorithms using techniques from the theory of Markov Decision Processes.

In the past decades, there has been no significant progress in the development of online approximation algorithms for solving the k-CTP. Basically, two simple deterministic strategies are currently available [27,28]. The first is the *greedy* algorithm (GA), which starts at a vertex v and finds the shortest v-t path by using Dijkstra's algorithm [8] in a greedy manner based on the current blockage information. The second strategy, called the *reposition* algorithm (RA), proposed by Westphal [27], requires the traveller to begin at the source s and follow the shortest s-t path until he/she learns about a blockage on the path to t. At that point, he/she returns to s and takes a new shortest s-t path based on the updated blockage information. In addition, Westphal proved that 1) no deterministic online algorithm within a $(2k+1)$-competitive ratio exists for the problem; and 2) the simple reposition algorithm can achieve the lower bound. He also proved a lower bound of $k+1$ on the competitive ratios of all randomized online algorithms. Xu et al. [28] 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. Liao and Huang [15] 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 design a tour through a set of vertices [16], where the traveller visits each vertex and returns to the origin under the same uncertainty. This problem is similar to

the online Traveling Salesman Problem (TSP) and its variations [14,23]. While deterministic algorithms have been extensively studied, there was a dearth of research on randomized approaches for solving the problem. Recently, Bender and Westphal proposed a randomized algorithm for special graphs in which all s-t paths are vertex-disjoint [3]. There were also some recent work on the k-CTP for special graphs [4,5,24] and the variants of the k-CTP [11,25,29]. Note that most of the algorithms are based on the RA strategy.

Our results. In this paper, we have developed randomized strategies for solving the k-CTP. We have proposed 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 the algorithm can be improved to $\left(1+\frac{\sqrt{2}}{2}\right)k+\sqrt{2}$ 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 as follows. Given a connected edge-weighted graph $G=(V,E)$ with a source s and a destination t in V and a distance function $d: E \rightarrow R^+$, 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. More precisely, the set comprises all s-t paths of cost $(1+\alpha)d(s,t)$, where $d(s,t)$ denotes the shortest travel time 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 removing 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 a similar argument until the traveller arrives at the destination t.

Near-shortest paths. To implement 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 on 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 $G = (V, A)$, takes constant time for each of the ℓ paths after a fast preprocessing step that runs in $O(|V| \log |V| + |A|)$ time [9,10]. 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 are the same distance. In Eppstein's study, cycles of repeated vertices were allowed. Recently, Carlyle and Wood [7], Hershberger et al. [13], and Frieder and Roditty [12] studied finding the ℓ shortest simple (i.e., loopless) paths in directed graphs. In addition, Katoh et al. [19] 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 currently the best known result for the problem in undirected graphs.

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, $1 \leq j \leq \mu$, where μ is at most the sum of the distances of all the 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 $d^1(s,t) = d(s,t)$. The technique can represent all strictly jth-shortest paths of cost $d^j(s,t)$, $1 \leq j \leq \mu$. 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, in the worst-case, the number of such near-shortest s-t paths may be exponential in the order of G. The proposed implicit representation can guarantee the pseudo-polynomial running time of the randomized routing strategy.

2 Preliminaries

In this paper, we consider the k-CTP in which the number of blockages is bounded by a given constant k. Given a connected edge-weighted 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. In the following we simply

use E_i instead of E_i^A if there is no confusion. Besides, we 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^A during the trip; and let $d_{E_k}(s,t)$ be the offline optimum from s to t under complete information E_k. For ease of convenience, we use *x-path* to denote a route on which the traveller spends at most x. For example, *$d_{E_i}(s,t)$-path* denotes a route on 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 [6,26] 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 [1,28]: 1) once a blocked edge is discovered by the traveller, the edge remains blocked permanently; and 2) the given connected graph G remains connected even if all the blocked edges are eliminated.

Recall the two deterministic routing strategies GA and RA [27,28]. When the traveller starts his/her journey at some vertex v, the distance cost of the shortest v-t path derived by GA is $d_{E_i}(v,t)$ under blockage information E_i. If all the k blocked edges are known at the outset, then the cost of the path obtained by GA from the source s is the same as the offline optimum $d_{E_k}(s,t)$. On the other hand, for a single blockage e_i discovered by the RA strategy, the travel cost is at most $2d_{E_i}(s,t)$ under blockage information E_i. Therefore, RA can derive the deterministic lower bound of $2k+1$ for k blockages.

The remainder of this paper is organized as follows. In Section 3, we analyze the competitive ratio of our main algorithm based on the assumption that the randomized strategy can help the traveller traverse an apex tree T efficiently; and in Section 4, we describe the randomized strategy for traversing the apex tree T. In Section 5, we present a simple implicit representation of multiple near-shortest s-t paths; or more precisely, shortest to strictly μth-shortest s-t paths for a sufficiently large value of μ. Section 6 contains our concluding remarks.

3 Main Algorithm

Given a connected graph $G = (V, E)$ with a source s and a destination t, here 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 small constant, $0 < \alpha < 1$. In contrast to previous studies, we find strictly jth-shortest paths, $1 \leq j \leq \mu$, for a sufficiently large value of μ to derive the set S of the s-t paths. We discuss the technique for doing this in Section 5.

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
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;
  $\frac{k-i}{k-i+1}$: proceeding to t along an arbitrary $d_{E_i}(s,t)$-path until a blockage is found;
  $\frac{1}{k-i+1}$: following **procedure Traverse-Tree** on the remaining s-t paths in S;
  Let the number of blockages discovered by the traveller be j;
  if the traveller has not arrived at t **then**
    if $j < k$ **then**
      the traveller returns to s and $i \leftarrow j$;
    else if $j = k$ **then** ▷ find all blockages
      the traveller returns to s and follows a $d_{E_k}(s,t)$-path to t;
    end if
  end if
while (the traveller has not arrived at t)

The steps of the main algorithm (see Algorithm 1) are 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, $0 \leq i \leq k$, for a small α. Then, the traveller uses the reposition RA strategy, and when restarting at s, he/she 1) selects an arbitrary $d_{E_i}(s,t)$-path from S and traverses the path with probability $\frac{k-i}{k-i+1}$; or 2) chooses the remaining s-t paths in S and traverses them using the Traverse-Tree procedure with probability $\frac{1}{k-i+1}$. This randomized strategy finds a balance between these two operations in an online fashion. That is, the second operation could compensate for some loss if we always use the RA strategy. Note that the traveller may learn about more than one blocked edge while performing the operations in Line 4. The algorithm repeats until the traveller arrives at t.

To analyze the competitive ratio of the entire *GRR* algorithm, the traveller must be able to efficiently traverse the remaining s-t paths in S by following the Traverse-Tree procedure, which we explain 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 $2k+1$. We also prove the correctness of the claim 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, $0 \leq i < k$, then each blockage discovered during the trip will increase the total travel cost of the algorithm by at most $(1+\alpha)d_{E_i}(s,t)$ on average.

Based on the above claim, we derive the following theorem.

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

Proof We divide the proof into two cases, depending on if the traveller uses a $d_{E_k}(s,t)$-path to t (as indicated in Line 10 of the *GRR* algorithm). That is, 1) the traveller exploits RA or the Traverse-Tree procedure to reach t in an apex tree, or 2) the Traverse-Tree procedure fails in possibly many iterations of the do-while loop and the traveller eventually follows a $d_{E_k}(s,t)$-path to t. In

the worst-case scenario, suppose the traveller learns about only one blockage in each iteration of the loop. Note that in Case 1, the traveller may learn k' blockages, $k' \leq k$, while in Case 2, all the k blockages are discovered. Thus, in every iteration r, $r \geq 1$, assume the number of remaining undiscovered blockages is $k' - r + 1$ in Case 1 ($k - r + 1$ in Case 2). We let the cost of implementing the Traverse-Tree procedure be $c(r)$ for the blockages discovered in iteration r, while the cost of performing RA is at most $2d_{E_{r-1}}(s,t)$. In the following, we consider the worst case by letting $k' = k$.

Case 1: The traveller arrives at t by using RA or the Traverse-Tree procedure in an apex tree; that is, at least one $(1+\alpha)d_{E_i}(s,t)$-path from s to t is unblocked during the trip, where $0 \leq i < k$. Notice that $(1+\alpha)d_{E_i}(s,t) > d_{E_k}(s,t)$ possibly occurs in the *GRR* algorithm.

Based on Claim 1, we have $c(r) \leq (k-r+1)(1+\alpha)d_{E_i}(s,t)$ for some i, $i \leq k-1$ in every iteration r of the loop. Consequently, the expected travel cost of the *GRR* algorithm is formulated as follows:

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

As mentioned earlier, if $(1+\alpha)d_{E_{k-1}}(s,t) > d_{E_k}(s,t)$ occurs, then the last inequality is directly obtained. Otherwise, it leads to a simpler form in terms of $d_{E_k}(s,t)$. Thus, in Case 1, due to $d_{E_{k-1}}(s,t) \leq d_{E_k}(s,t)$, the competitive

ratio of the GRR algorithm is at most

$$\frac{\left(k+\frac{1}{2}(1+\alpha)k+(1+\alpha)\right)\cdot d_{E_{k-1}}(s,t)}{d_{E_k}(s,t)} \leq \frac{3+\alpha}{2}\cdot k+(1+\alpha).$$

Case 2: The traveller fails to use RA or the Traverse-Tree procedure to reach t in an apex tree and thus Claim 1 cannot be applied. That is, he/she eventually restarts at s and follows a $d_{E_k}(s,t)$-path to t, as indicated in Line 10 of the GRR algorithm. Precisely, every $(1+\alpha)d_{E_{k-1}}(s,t)$-path is blocked during the trip, which implies that the distance of an offline optimal s-t path is $d_{E_k}(s,t) > (1+\alpha)d_{E_{k-1}}(s,t)$.

Therefore, an upper bound on the cost $c(r)$ of implementing the Traverse-Tree procedure in iteration r can be derived by exploiting the RA strategy [27]. That is, $c(r) \leq (k-r+1)\cdot 2(1+\alpha)d_{E_{k-1}}(s,t)$ for every iteration r. The expected total travel cost of the GRR algorithm is formulated in a similar manner as follows:

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

Hence, in Case 2, the 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 $\sqrt{2}-1$. Consequently, the expected competitive ratio is at most $\left(1+\frac{\sqrt{2}}{2}\right)k+\sqrt{2}$. Note that for any small constant α', $0<\alpha'\leq\alpha=\sqrt{2}-1$, the competitive ratio is still smaller than the deterministic lower bound of $2k+1$ when $k\geq 2$. □

4 Apex Tree

In this section, we consider the Traverse-Tree procedure implemented 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.

We refer to the definition of an *apex graph* in planar graph theory and have the following definition. A graph $T = (V, E)$ is an *apex tree* if T contains a source vertex s, a rooted tree that comprises 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 and there is exactly one path from each vertex to t in $T \setminus \{s\}$.

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, reported in [27], of establishing the lower bound of $k+1$ is also an apex tree in which all s-t paths have the same cost. Thus, the competitive ratio of the algorithm can achieve the lower bound, and it follows that the algorithm is optimal.

Algorithm 2: Traverse-Tree Procedure

Input : An apex tree T that represents s-t paths and a constant k;
Output : A random route from s to t, or the traveller fails to reach t;

Assign equal probabilities among the children of the root t, and sequentially repeat the process for each descendant of t in the order of a breadth-first search;
The traveller begins at s, and randomly selects an s-t path based on the assigned probability and proceeds to t on that path;
while the traveller does not arrive at t and there is still an available s-t route **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 ignores the blocked s-v_i path;
    while every s-t path through the vertex v_{i+1} is currently blocked and $i+1 \leq h$ **do**
        $i \leftarrow i+1$; ▷ depth-first search order of the path p
    end while
    if $i+1 \leq h$ **then** ▷ there is still an s-t path; otherwise, the traveller cannot reach t
        Reassign probabilities to the subtree that is rooted at v_{i+1} in a similar way;
        The traveller randomly selects an s-t path through v_{i+1} based on the assigned probability and proceeds to t on that path;
    end if
end while

The main concept of the Traverse-Tree procedure is to incorporate randomized operations into the reposition RA strategy and then explore subtrees of an apex tree T in the order of a *depth-first search*. More precisely, we initially distribute the probabilities of path selection equally among the children of the root t. Next, we sequentially distribute the probabilities equally among the descendants in the order of a *breadth-first search*. When the traveller starts at the source s, he/she randomly selects an s-t path according to the assigned probability and follows the path to t in the apex tree T. If the traveller finds a blockage on the way to t, he/she uses the RA strategy and returns to s. We ignore the blocked path, and reassign the probabilities to the unblocked subtrees in a similar manner. The traveller traverses the remaining routes in T by exploring the subtrees in the order of a *depth-first search*. The argument is repeated until the traveller arrives at the destination t.

Figure 1 shows an example with $k = 3$ that illustrates the steps of the Traverse-Tree procedure for the k-CTP. The traveller starts at the source s and randomly selects an s-t path based on the initial probabilities, as shown in Figure 1(a). Suppose the traveller traverses the s-t path $s - v_2 - v_4 - v_7 - t$ with probability $\frac{1}{12}$ until he/she finds a blocked edge (v_2, v_4) at v_2. He/she then returns to s and randomly selects an s-t path through v_4 if any exists; accordingly, he/she selects either $s - v_1 - v_4 - v_7 - t$ or $s - v_3 - v_4 - v_7 - t$ (Figure 1(b)). If the next blockage occurs in (v_4, v_7) upon arrival at v_4, the traveller has exactly one s-t path via v_7, which is $s - v_5 - v_6 - v_7 - t$, so he/she returns to s and follows the path directly (Figure 1(c)). The last blocked edge discovered by the traveller is (v_7, t) when he/she reaches v_7. The traveller returns to s and randomly selects either $s - v_8 - v_{10} - v_{11} - t$ or $s - v_9 - v_{10} - v_{11} - t$ (Figure 1(d)).

For the k-CTP in an apex tree T, there is at least one s-t path without a blockage; that is, the offline optimal s-t path. Let the offline optimal s-t path be $p : s = v_1^* - v_2^* - \cdots - v_m^* = t$; and let the number of children of a vertex v_j^* in the apex tree T be c_j, such that v_j^* has children $v_{j,1}, v_{j,2}, \ldots, v_{j,c_j}$. Without loss of generality, assume the last child of each v_j^*, v_{j,c_j}, $2 \leq j \leq m$,

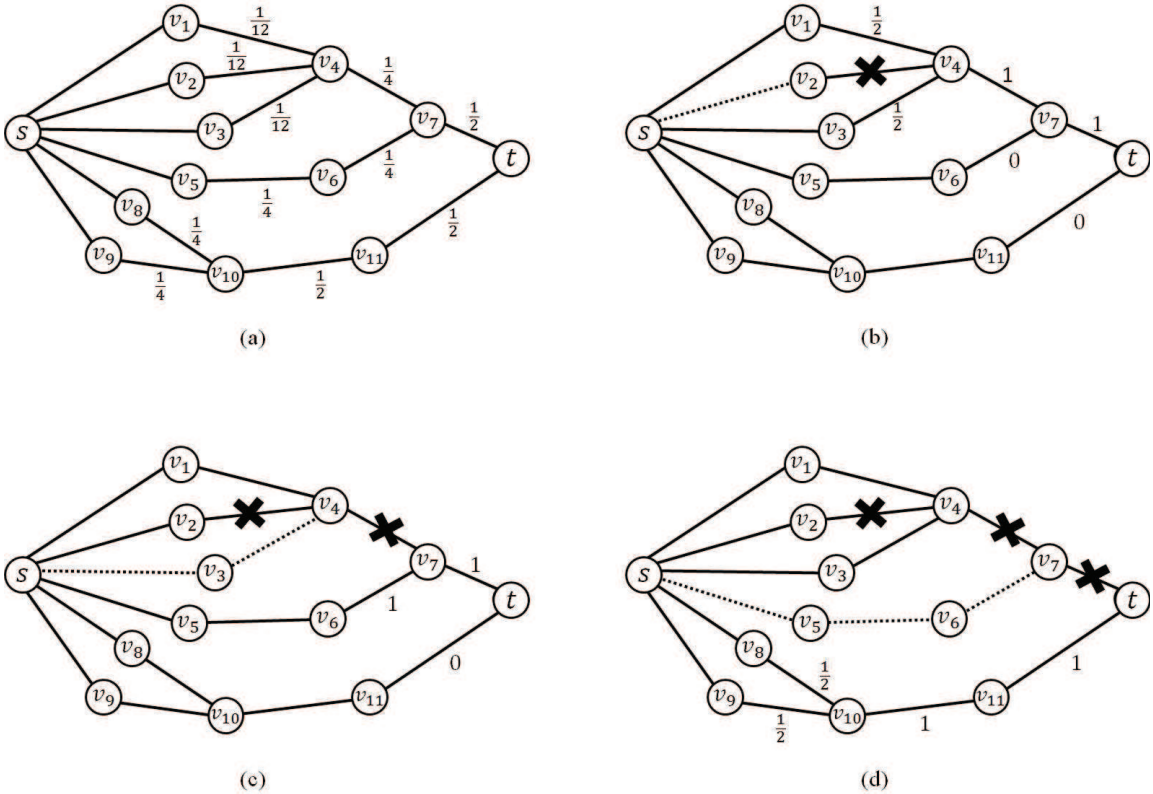

Fig. 1 An example of the Traverse-Tree procedure

lies on the path p; that is, $v_{j,c_j} = v^*_{j-1}$. In addition, suppose each subtree that is rooted at $v_{j,\ell}$, $1 \leq \ell \leq c_j - 1$, has $b_{j,\ell}$ blockages. Consider the expected total cost when the traveller uses the algorithm to traverse paths other than the offline optimal path. Note that the malicious adversary does not block any edge $(s, v) \in E$. Thus, $\sum_{j=2}^{m} \sum_{\ell=1}^{c_j-1} b_{j,\ell} \leq k$. That is, each blocked edge is counted at most once when considering the subtrees rooted at each of the children of every vertex along the offline optimal path (except those vertices in the path themselves). We are ready to prove the following lemma.

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

Proof We prove the statement in a top-down manner for an apex tree T. Notice that the length of the offline optimal s-t path in T is the same as that of every other s-t path. 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 $\mathrm{E}(s, v_m^*) \leq \mathrm{E}(s, v_{m-1}^*) + d_{E_k}(v_{m-1}^*, v_m^*)$. If $c_m > 1$, the traveller finds v_{m-1}^* as a predecessor of v_m^* with probability $\frac{1}{c_m}$ 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, he/she will find it in the second trial with probability $\left(1 - \frac{1}{c_m}\right)\frac{1}{c_m - 1}$. Suppose the traveller selects v_{m,i_j}, $1 \leq i_j \leq c_m - 1$ in a random order, when trying to find $v_{m-1}^* = v_{m,c_m}$. That is, without loss of generality, he/she selects the next one from v_{m,i_1} to $v_{m,i_{c_m-1}}$. In this case, the expected travel cost is at most $\left\{2b_{m,i_1} d_{E_{b_{m,i_1}}}(s, v_{m,i_1}) + \mathrm{E}(s, v_{m-1}^*) + d_{E_k}(v_{m-1}^*, v_m^*)\right\}$. This is because RA may return to s b_{m,i_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 $d_{E_k}(v_{m-1}^*, v_m^*)$. We obtain

$$
\begin{aligned}
\mathrm{E}(s, v_m^*) \leq{}& \frac{1}{c_m}\left\{\mathrm{E}(s, v_{m-1}^*) + d_{E_k}(v_{m-1}^*, v_m^*)\right\} \\
&+ \left(1 - \frac{1}{c_m}\right)\frac{1}{c_m - 1}\left\{2b_{m,i_1} d_{E_{b_{m,i_1}}}(s, v_{m,i_1}) + \mathrm{E}(s, v_{m-1}^*) + d_{E_k}(v_{m-1}^*, v_m^*)\right\} \\
&+ \left(1 - \frac{1}{c_m}\right)\left(1 - \frac{1}{c_m - 1}\right)\frac{1}{c_m - 2}\left\{2b_{m,i_1} d_{E_{b_{m,i_1}}}(s, v_{m,i_1}) + 2b_{m,i_2} d_{E_{b_{m,i_2}}}(s, v_{m,i_2})\right. \\
&\left. + \mathrm{E}(s, v_{m-1}^*) + d_{E_k}(v_{m-1}^*, v_m^*)\right\} \\
&+ \cdots \\
&\leq \frac{2}{c_m}\left\{((c_m - 1)b_{m,i_1} + (c_m - 2)b_{m,i_2} + \cdots + b_{m,i_{c_m-1}}) d_{E_k}(s, t)\right\} \\
&+ \mathrm{E}(s, v_{m-1}^*) + d_{E_k}(v_{m-1}^*, v_m^*) \\
&= \frac{2}{c_m}\left\{\frac{c_m(c_m - 1)}{2}\left(\frac{1}{c_m - 1}\sum_{\ell=1}^{c_m - 1} b_{m,\ell}\right) d_{E_k}(s, t)\right\} + \mathrm{E}(s, v_{m-1}^*) + d_{E_k}(v_{m-1}^*, v_m^*) \\
&= \sum_{\ell=1}^{c_m - 1} b_{m,\ell} d_{E_k}(s, t) + \mathrm{E}(s, v_{m-1}^*) + d_{E_k}(v_{m-1}^*, v_m^*)
\end{aligned}
$$

Therefore, irrespective of whether $c_m = 1$, we obtain $\mathrm{E}(s, v_m^*) \leq \sum_{\ell=1}^{c_m - 1} b_{m,\ell} d_{E_k}(s, t) + \mathrm{E}(s, v_{m-1}^*) + d_{E_k}(v_{m-1}^*, v_m^*)$. Similarly, we obtain $\mathrm{E}(s, v_{m-1}^*) \leq \sum_{\ell=1}^{c_{m-1} - 1} b_{m-1,\ell} d_{E_k}(s, t) + \mathrm{E}(s, v_{m-2}^*) + d_{E_k}(v_{m-2}^*, v_{m-1}^*)$. This implies that $\mathrm{E}(s, v_m^*) \leq \sum_{j=2}^{m} \sum_{\ell=1}^{c_j - 1} b_{j,\ell} d_{E_k}(s, t) + d_{E_k}(v_1^*, v_2^*) + d_{E_k}(v_2^*, v_3^*) + \cdots + d_{E_k}(v_{m-1}^*, v_m^*) \leq (k+1) d_{E_k}(s, t)$. Consequently,

the expected total cost of the algorithm (including the distance cost of the offline optimal path) is at most $(k+1)d_{E_k}(s,t)$. Hence, the competitive ratio can achieve the lower bound for randomized online algorithms in apex trees. □

Note that the lemma also holds if some blockages are not discovered by the Traverse-Tree procedure, i.e. $\sum_{j=2}^{m}\sum_{\ell=1}^{c_j-1} b_{j,\ell} = k' < k$. Precisely, the total expected cost is at most $(k'+1)d_{E_k}(s,t)$. Moreover, the result can be extended to a more general apex tree T in which the distance cost of each s-t path in T is at most $(1+\alpha)d_{E_i}(s,t)$, $0 \le i < k$. That is, the upper bound on the total expected travel cost of the procedure is $\sum_{j=2}^{m}\sum_{\ell=1}^{c_j-1} b_{j,\ell}(1+\alpha)d_{E_i}(s,t) + d_{E_k}(v_1^*, v_m^*) \le k(1+\alpha)d_{E_i}(s,t)+d_{E_k}(s,t)$. Note that $d_{E_k}(s,t) \le (1+\alpha)d_{E_i}(s,t)$ because every s-t path in T is bounded by $(1+\alpha)d_{E_i}(s,t)$. The competitive ratio is $(k(1+\alpha)d_{E_i}(s,t) + d_{E_k}(s,t))/d_{E_k}(s,t) = (1+\alpha)k+1$. It proves the claim made in Section 3, i.e., each blockage increases the total travel cost by at most $(1+\alpha)d_{E_i}(s,t)$ on average in the Traverse-Tree procedure. The next theorem follows immediately.

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 $(1+\alpha)k+1$.*

Recall the proof of our main result, Theorem 1. Claim 1 is applied when the traveller reaches t by using RA or the Traverse-Tree procedure in an apex tree T, irrespective of whether every blockage has been discovered. In fact, the traveller should follow a $(1+\alpha)d_{E_i}(s,t)$-path to t in the apex tree represented by a set of $(1+\alpha)d_{E_i}(s,t)$-paths, where $i \le k-1$. Precisely, when all the k blockages have been discovered, the $(1+\alpha)d_{E_i}(s,t)$-path is actually the $d_{E_k}(s,t)$-path. If the traveller fails to reach t in an apex tree, he/she follows a $d_{E_k}(s,t)$-path to t. That is, the distance cost of the last trial/path cannot be bounded by Claim 1.

We also remark that when the Traverse-Tree procedure performs, it does no need to know about the number of blockages, similar to the RA strategy.

Furthermore, the procedure can achieve the optimal competitive ratio in apex trees, irrespective of whether it finds all k blockages, which can guarantee the correctness of Theorem 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 provide two simple data structures for storing all the shortest s-t paths, strictly second-shortest s-t paths, to strictly μth-shortest s-t paths, for a large value of μ. More precisely, we set μ to be the sum of the distances of all edges in order to derive a sufficient number of near shortest s-t paths. 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 is for representing possibly non-simple paths whose distances are between the cost $d(s,t)$ and the cost $d^\mu(s,t)$.

Recall the definition of *strictly* jth-shortest s-t paths, $1 \leq j \leq \mu$, where μ is at most the sum of the distances of all the edges. That is, let $d^1(s,t), d^2(s,t), d^3(s,t), \ldots$ denote the strictly increasing sequence of all possible distinct s-t path weights, where $d^1(s,t) = d(s,t)$. Our technique can represent all strictly jth-shortest paths of cost $d^j(s,t)$, $1 \leq j \leq \mu$.

5.1 Strictly second-shortest paths

First, we define some notations. For each vertex $v \neq s$, let $S^j(v)$ be the vertex set that comprises all of v's predecessors, each of which is v's preceding neighbor lying on 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. In the graph, 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 in V^j are eliminated. Previous studies [17, 20, 21] have investigated the strictly second-shortest path problem (i.e., *next-to-shortest path*) based on the *shortest path digraph*, i.e., $D^1(G) = (V^1, A^1)$.

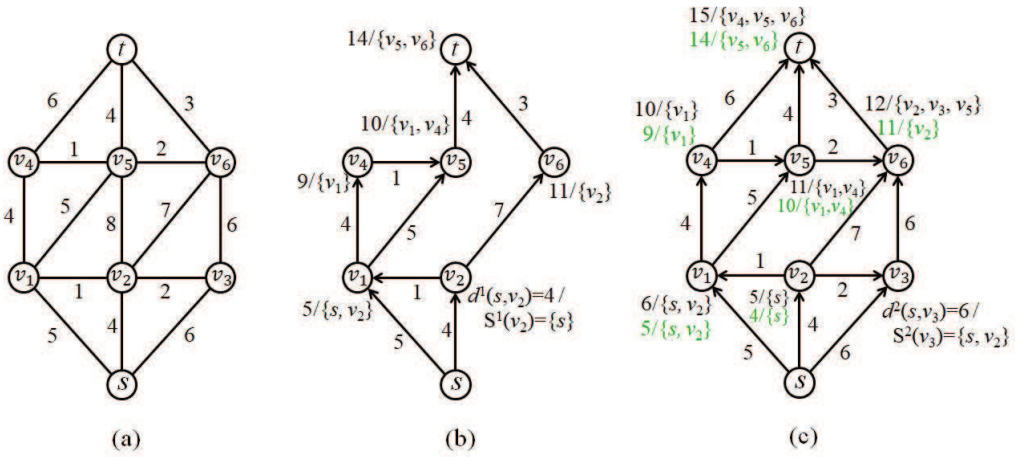

Fig. 2 (a) An instance graph G; (b) the shortest path digraph $D^1(G)$ with $d^1(s,t) = 14$ in which for each pair associated with a vertex v, the first number corresponds to $d^j(s,v)$, and the second to $S^j(v)$, where $j = 1$; (c) the second-shortest path digraph $D^2(G)$ with $d^2(s,t) = 15$ for storing all shortest to strictly second-shortest s-t paths

Notably, for any graph G, $D^1(G)$ is acyclic and can be constructed in $O(|V|^2)$ time [17,20]. Additionally, for every vertex $v \neq s$, $d^1(s,v)$ and $S^1(v)$ can be obtained.

Figure 2(b) shows an example of the shortest path digraph of a given graph. Based on the key property below, Kao et al.'s algorithm [17] can derive the cost of a strictly second-shortest s-t path, $d^2(s,t)$, and construct such a path in $O(|V|^2)$ time for a graph G.

Proposition 1 [17,21] *For each strictly second-shortest s-t path p, there is at least one edge $e = \overrightarrow{(u,v)} \notin A^1$ in p such that the s-u subpath and v-t subpath of p are exactly the shortest s-u path and the shortest v-t path respectively.*

We design the Find-2nd-Shortest procedure to search for all strictly second-shortest simple s-t paths and construct the representation $D^2(G) = (V^2, A^2)$. The main step of the procedure is simply a breadth-first search. We start at t and traverse all other vertices backward until s, and determine whether each vertex lies on a strictly second-shortest s-t path. Figure 2(c) shows the representation $D^2(G)$ of the instance graph. As shown in the figure, there is one type of strictly second-shortest path of cost 15: $s - \ldots - v_4 - t$ because

1: **procedure Find-2nd-Shortest**$(G, s, t, D^1(G))$ ▷ find each $d^2(s,v)$ and $S^2(v)$ in $D^2(G)$
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** every w adjacent to u in G and $\overrightarrow{(u,w)} \notin A^2$ **do** ▷ breadth-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** ▷ add w into $D^2(G)$
$d^2(s,u) - d(w,u) = d^1(s,w)$ and $w \in V \setminus V^1$ **then** ▷ update $d^2(s,w)$
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**

$v_4 \in S^2(t) \setminus S^1(t)$. Additionally, there are two other types of strictly second-shortest paths: $s - \ldots - v_3 - v_6 - t$ and $s - \ldots - v_5 - v_6 - t$ because v_3, $v_5 \in S^2(v_6) \setminus S^1(v_6)$.

The correctness of the procedure follows from Proposition 1 and the optimal substructure property. That is, $\overrightarrow{(w,u)}$ is included in A^2 in $D^2(G)$ (and w is put into $S^2(u)$) if the shortest s-w path plus $\overrightarrow{(w,u)}$ can construct a strictly second-shortest s-u path. 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. We duplicate each vertex that has multiple outgoing edges when assigning probabilities from t to its descendants. As the traveller selects a random path from s to t, he/she excludes the vertices already visited in previous iterations. Thus, Algorithm 2 can work in $D^2(G)$.

In addition, the procedure can be generalized to find 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 Proposition 1 holds between every strictly $(j-1)$th-shortest path and strictly jth-shortest path. However, Kao et al.'s method cannot be extended straightforwardly to compute $d^j(s,t)$, $j \geq 3$. (A brute-

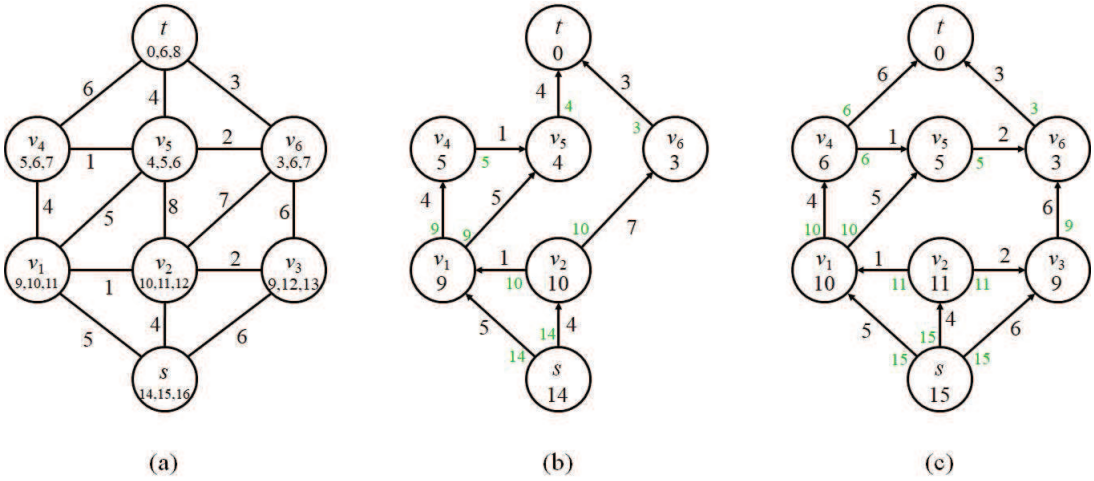

Fig. 3 (a) An example of the data structure for storing up to strictly third-shortest s-t paths; (b) the graph for storing the shortest s-t paths of cost 14; (c) the graph for storing s-t paths of cost 15

force approach whose running time is exponential in j is intuitive.) We leave finding an efficient way to derive $d^j(s,t)$, $j \geq 3$ as an open problem.

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 at most $d^2(s,t) = (1+\alpha')d(s,t)$ for some $\alpha' > 0$. If $\alpha' < \alpha = \sqrt{2}-1$, the competitive ratio would be $\max\{(\frac{3+\alpha'}{2})k + (1+\alpha')\ ,\ (\frac{2+\alpha'}{1+\alpha'})k+1\} < 2k+1$. Otherwise, $\alpha' \geq \alpha$ leads to a better competitive ratio when we just use the RA strategy on the remaining s-t paths. Therefore, the *GRR* algorithm can improve the deterministic lower bound for the k-CTP in polynomial time.

Theorem 3 *The GRR algorithm can approximate the* k-Canadian Traveller Problem *with a competitive ratio less than* $2k+1$ *in polynomial time, by using the* Find-2nd-Shortest *procedure to store the set of near shortest s-t paths.*

5.2 Strictly μth-shortest paths

The steps of the algorithm for finding possibly non-simple strictly μth-shortest paths are as follows. To compute the sets of strictly jth-shortest paths for $j = 1, 2, \ldots, \mu$, we use the Find-Multiple-Shortest procedure, which is also a breadth-first search that traverses backward from t to s. Next, we define some notations. In the ith iteration, we keep a set of vertices S_i, which are reached from t using exactly i edges; and $S_0 = \{t\}$ initially. Let a set $D_V(u)$ store the shortest to strictly μth-shortest distances from u to t; that is, $D_V(s) = \{d^1(s,t), d^2(s,t), \ldots, d^\mu(s,t)\}$. In addition, let two sets $D_E(\overrightarrow{(u,v)})$ and $D_E(\overrightarrow{(v,u)})$ for an edge $e = (u,v)$ store the distances from u to t and the distances from v to t respectively, using the edge e. More precisely, $\ell \in D_E(\overrightarrow{(u,v)})$ for an edge $e = (u,v)$ if and only if there is a path of distance ℓ, starting at u and passing through the edge e to t; i.e., ℓ is the shortest to strictly μth-shortest distances from u along e to t. Note that the breadth-first search traverses a vertex multiple times. It is sufficient to repeat the iteration $\mu|E|$ times because a strictly μth-shortest s-t path uses at most $\mu|E|$ edges. Figure 3(a) shows an example of the data structure that represents all the shortest to strictly third-shortest s-t paths in the instance graph G in Figure 2(a). In the graph, the set of numbers associated with a vertex u corresponds to $D_V(u)$.[1]

Based on the above data structure, we can exploit the Implicit-Representation procedure to construct a graph $G' = (V', E')$ that represents all (possibly non-simple) s-t paths whose distances are in a given set $L \subseteq \{d^1(s,t), \ldots, d^\mu(s,t)\}$.

[1] Due to space limit, the value of each $D_E(\overrightarrow{(u,v)})$ is shown: $D_E(\overrightarrow{(s,v_1)}) = \{14,15,16\}$, $D_E(\overrightarrow{(s,v_2)}) = \{14,15,16\}$, $D_E(\overrightarrow{(s,v_3)}) = \{15,18,19\}$; $D_E(\overrightarrow{(v_1,s)}) = \{19,20,21\}$, $D_E(\overrightarrow{(v_1,v_2)}) = \{11,12,13\}$, $D_E(\overrightarrow{(v_1,v_4)}) = \{9,10,11\}$, $D_E(\overrightarrow{(v_1,v_5)}) = \{9,10,11\}$; $D_E(\overrightarrow{(v_2,s)}) = \{18,19,20\}$, $D_E(\overrightarrow{(v_2,v_1)}) = \{10,11,12\}$, $D_E(\overrightarrow{(v_2,v_3)}) = \{11,14,15\}$, $D_E(\overrightarrow{(v_2,v_5)}) = \{12,13,14\}$, $D_E(\overrightarrow{(v_2,v_6)}) = \{10,13,14\}$; $D_E(\overrightarrow{(v_3,s)}) = \{20,21,22\}$, $D_E(\overrightarrow{(v_3,v_2)}) = \{12,13,14\}$, $D_E(\overrightarrow{(v_3,v_6)}) = \{9,12,13\}$; $D_E(\overrightarrow{(v_4,v_1)}) = \{13,14,15\}$, $D_E(\overrightarrow{(v_4,v_5)}) = \{5,6,7\}$, $D_E(\overrightarrow{(v_4,t)}) = \{6,12,14\}$; $D_E(\overrightarrow{(v_5,v_1)}) = \{14,15,16\}$, $D_E(\overrightarrow{(v_5,v_2)}) = \{18,19,20\}$, $D_E(\overrightarrow{(v_5,v_4)}) = \{6,7,8\}$, $D_E(\overrightarrow{(v_5,v_6)}) = \{5,8,9\}$, $D_E(\overrightarrow{(v_5,t)}) = \{4,10,12\}$; $D_E(\overrightarrow{(v_6,v_2)}) = \{17,18,19\}$, $D_E(\overrightarrow{(v_6,v_3)}) = \{15,18,19\}$, $D_E(\overrightarrow{(v_6,v_5)}) = \{6,7,8\}$, $D_E(\overrightarrow{(v_6,t)}) = \{3,9,11\}$; $D_E(\overrightarrow{(t,v_4)}) = \{11,12,13\}$, $D_E(\overrightarrow{(t,v_5)}) = \{8,9,10\}$, $D_E(\overrightarrow{(t,v_6)}) = \{6,9,10\}$. Note that for a directed edge (u,v), $D_E(\overrightarrow{(u,v)}) = \{\ell + d(u,v) \mid \ell \in D_V(v)\}$.

```
1: procedure Find-Multiple-Shortest(G, s, t, μ)      ▷ find each D_V(u) and D_E(⟨u, v⟩)
2:     Duplicate each edge in the input graph G = (V, E) to make G directed;
3:     Let S_0 = {t};
4:     Let D_V(v) = {∞}, ∀v ∈ V, and D_V(t) = {0};
5:     Let D_E(⟨u, v⟩) = {∞} for each edge e = ⟨u, v⟩ ∈ E;
6:     for i = 0 to μ|E| do
7:         S_{i+1} ← ∅;
8:         for each v ∈ S_i do
9:             for each e = ⟨u, v⟩ ∈ E do
10:                S_{i+1} ← S_{i+1} ∪ {u};        ▷ Let L comprise possible distances from u to t
11:                L ← {d(u, v) + ℓ | ℓ ∈ D_V(v)};
12:                Let D_E(⟨u, v⟩) be the set of the smallest μ values in D_E(⟨u, v⟩) ∪ L;
13:            end for
14:        end for
15:        for each u ∈ V do
16:            for each e = ⟨u, v⟩ ∈ E do
17:                Let D_V(u) be the set of the smallest μ values in D_V(u) ∪ D_E(⟨u, v⟩);
18:            end for
19:        end for
20:    end for
21: end procedure
```

Starting from s, we only traverse an edge $e = (u, v)$ if the set $D_E(\overrightarrow{(u, v)})$ of distances to t contains the given set of distances. Similarly, in the ith iteration, we keep a set of vertices S_i, which are reached from s using exactly i edges; and $S_0 = \{s\}$ initially. Let $L(v)$ be the set of distances from v to t for each vertex $v \in V'$. More 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 in the procedure is also at most $\mu|E|$. As shown in Figures 3 (b) and (c), the procedure constructs two graphs, which represent all the shortest s-t paths of distance cost 14 and all strictly second-shortest s-t paths of cost 15, respectively. Similarly, the set of numbers associated with a vertex u corresponds to $D_V(u)$, and the set of green numbers associated with an edge $e = (u, v)$ corresponds to $D_E(\overrightarrow{(u, v)})$. Note that 1) the second-shortest path digraph $D^2(G)$ in Figure 2(c) merges the two graphs; and 2) the representation graph in Figure 3(b) is actually the same as the shortest path digraph $D^1(G)$ in Figure 2(b).

The resulting graph G' is a directed graph from s to t. However, because there may exist non-simple paths, the graph may contain cycles, as shown in Figure 4(a). The graph represents all the s-t paths of cost 16. There are

```
1: procedure Implicit-Representation(G, s, t, L)   ▷ construct G' = (V', E') to store
   all s-t paths whose distances are in L
2:     Initially, i = 0, S_i = {s}, V' = {s}, E' = ∅, L(v) = ∅, ∀v ∈ V, and L(s) = D_V(s) ∩ L;
3:     while S_i ≠ ∅ do
4:         S_{i+1} ← ∅;
5:         for each u ∈ S_i do
6:             for each e = (u, v) ∈ E do
7:                 L ← {ℓ − d(u, v) | ℓ ∈ L(u)};
8:                 L(v) ← L(v) ∪ (D_V(v) ∩ L);   ▷ find possible distances in L(v) using e
9:                 if L(v) ≠ ∅ then
10:                    S_{i+1} ← S_{i+1} ∪ {v} and V' ← V' ∪ {v};
11:                    E' ← E' ∪ {e};
12:                end if
13:            end for
14:        end for
15:        i ← i + 1;
16:    end while
17: end procedure
```

multiple edges between v_1 and v_2, and v_4 and v_5. 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' in Figure 4(a) is converted so that vertices v_1, v_2, v_4, v_5 are 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 the newly inserted vertices and edges becomes a directed acyclic graph, as shown in Figure 4(b). Therefore, the graph G'' can represent an apex tree, as described in Section 5.1.

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 sum of the distances of all edges. The number of iterations of the loop in the *GRR* algorithm is at most k, so the whole process takes $O(k\mu^2|E|^2)$ time in the worst case; that is, the proposed $\left(1 + \frac{\sqrt{2}}{2}\right)k + \sqrt{2}$-competitive randomized algorithm runs in pseudo-polynomial time.

Theorem 4 *The GRR algorithm can approximate the* k-Canadian Traveller Problem *within a competitive ratio of* $\left(1 + \frac{\sqrt{2}}{2}\right)k + \sqrt{2}$ *in pseudo-*

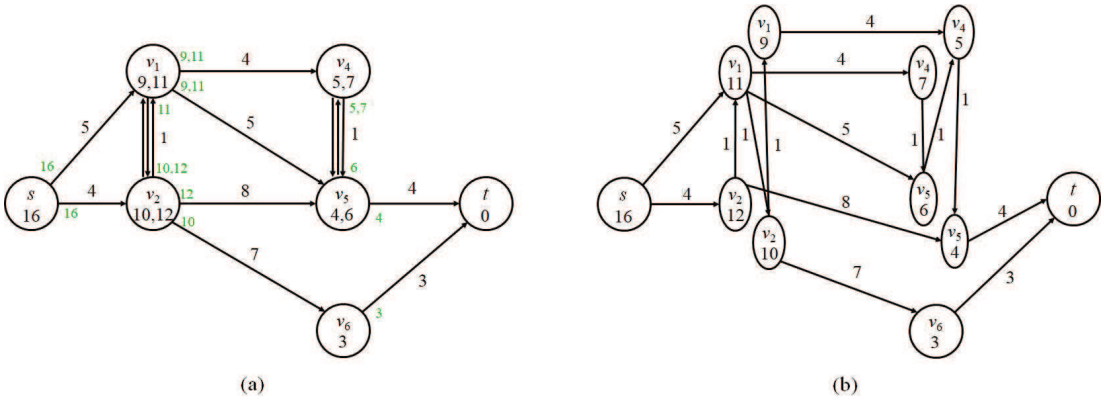

Fig. 4 The data structure for storing all s-t paths of distance cost 16

polynomial time, by using the Find-Multiple-Shortest *procedure to store the set of near shortest s-t paths.*

We remark that our simple implicit representations can be used to find the μth-best distinct values among all possible solutions for many optimization problems that can be expressed as shortest path problems. The applications are similar in spirit to finding the ℓ best solutions in Eppstein's study [9,10] for problems that can be solved by dynamic programming, such as the Knapsack problem and the sequence alignment problem.

6 Concluding Remarks

In this study, we have investigated the k-Canadian Traveller Problem when the number of blockages is up to a given constant k. Our major contribution is the first polynomial time randomized algorithm that can surpass the barrier of the deterministic lower bound of $(2k+1)$. The competitive ratio of the algorithm can be improved to $\left(1+\frac{\sqrt{2}}{2}\right)k+\sqrt{2}$ in pseudo-polynomial running time. Moreover, the proposed technique can be exploited to implicitly represent all strictly jth-shortest s-t paths, $1 \le j \le \mu$, in $O(\mu^2|E|^2)$ time and $O(\mu|E|)$ space.

We conclude this work by highlighting two open issues. First, it would be of great interest if all strictly jth-shortest paths could be computed more effi-

ciently. Second, the probability assignment of our randomized GRR algorithm is based on the known number of blockages. The issue concerns the difficulty a traveller may experience in using a randomized strategy without information about the number of blockages. Note that the reposition RA algorithm can reach the deterministic lower bound without knowing the number of blockages. In addition, the proposed Traverse-Tree procedure also works even if the number of blockages is unknown, and can achieve the lower bound for randomized algorithms in apex trees in which the distance cost of every s-t path is identical.

Acknowledgements The authors would like to thank anonymous referees for their helpful comments, as well as Dr. Fan Chung with UC San Diego, Dr. Kazuo Iwama with Kyoto University and Dr. Wing-Kai Hon with National Tsing Hua University for discussions on this work.

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 Combinatorial Optimization*, (2015), 30(1), pp. 87–96.
4. P. Bergé, J. Hemery, A. Rimmel, and J. Tomasik. The competitiveness of randomized strategies for Canadians via systems of linear inequalities. In *Proc. of 32nd International Symposium on Computer and Information Sciences (ISCIS)* (2018), Springer, pp. 96–103.
5. P. Bergé, J. Hemery, A. Rimmel, and J. Tomasik. On the competitiveness of memoryless strategies for the k-Canadian traveller problem. In *Proc. of 12th International Conference on Combinatorial Optimization and Applications (COCOA)* (2018), LNCS 11346, pp. 566–576.
6. A. Borodin and R. El-Yaniv. Online computation and competitive analysis. Cambridge University Press, Cambridge, (1998).
7. W.M. Carlyle and R.K. Wood. Near-shortest and k-shortest simple paths, *Networks*, (2005), 46(2), pp. 98–109.
8. E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, (1959), 1(1) pp. 269–271.
9. D. Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, (1998), 28(2), pp. 652–673.
10. D. Eppstein. k-Best enumeration. *Encyclopedia of Algorithms*, Springer, (2014), pp. 1–4.
11. D. Fried, S. E. Shimony, A. Benbassat, and C. Wenner. Complexity of Canadian traveler problem variants. *Theoretical Computer Science*, (2013), 487(27), pp. 1–16.
12. A. Frieder and L. Roditty. An experimental study on approximating k shortest simple paths. *ACM Journal of Experimental Algorithmics*, (2014), 19(1).
13. J. Hershberger, M. Maxel and S. Suri. Finding the k shortest simple paths: A new algorithm and its implementation. *ACM Transactions on Algorithms*, (2007), 3(4), No.45.
14. P. Jaillet and M.R. Wagner. Generalized online routing: New competitive ratios, resource augmentation, and asymptotic analyses. *Operations Research*, (2008), 56(3), pp. 745–757.

15. C.S. Liao and Y. Huang. Generalized Canadian traveller problems. *Journal of Combinatorial Optimization*, (2015), 29(4), pp. 701–712.
16. C.S. Liao and Y. Huang. The Covering Canadian traveller problem. *Theoretical Computer Science*, (2014), 530(17), pp. 80–88.
17. 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.
18. D. Karger and E. Nikolova. Exact algorithms for the Canadian traveller problem on paths and trees. Technical report, MIT Computer Science & Artificial Intelligence Lab, (2008).
19. N. Katoh, T. Ibaraki, and H. Mine. An efficient algorithm for k shortest simple paths. *Networks*, (1982), 12(4), pp. 411–427.
20. I. Krasikov and S.D. Noble. Finding next-to-shortest paths in a graph. *Information Processing Letters*, (2004), 92, pp. 117–119.
21. K.N. Lalgudi and M.C. Papaefthymiou. Computing strictly-second shortest paths. *Information Processing Letters*, (1997), 63, pp. 177–181.
22. C.H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, (1991), 84(1), pp. 127–150.
23. H.N. Psaraftis, M. Wen, and C.A. Kontovas. Dynamic vehicle routing problems: Three decades and counting. *Networks*, (2016), 67(1), pp. 3–31.
24. D. Shiri and F. S. Salman. On the randomized online strategies for the k-Canadian traveler problem. *Journal of Combinatorial Optimization*, (2019), Published Online, pp. 1–14.
25. D. Shiri and F. S. Salman. On the online multi-agent O-D k-Canadian Traveler Problem. *Journal of Combinatorial Optimization*, (2017), 34(2), pp. 453–461.
26. D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, (1985), 28, pp. 202–208.
27. S. Westphal. A note on the k-Canadian traveller problem. *Information Processing Letters*, (2008), 106, pp. 87–89.
28. 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 Combinatorial Optimization*, (2009), 18, pp. 195–205.
29. H. Zhang, Y.F. Xu, and L. Qin. The k-Canadian Travelers Problem with communication. *Journal of Combinatorial Optimization*, (2013), 26(2), pp. 251–265.