# A Survey of Randomized Algorithms for the Canadian Traveller Problem

Alek Westover[1], Nathan Sheffield[1], and Ryan Chang[1]

*alekw@mit.edu, shefna@mit.edu, rychang@mit.edu*
[1]Massachusetts Institute of Technology

December 2022

## Abstract

The Canadian Traveller Problem (CTP) is a widely studied variant of the shortest-path problem, in which a traveler discovers unexpected edge blockages in an online manner. We present known lower-bounds on competitive ratio for deterministic algorithms in terms of the number of blocked edges, then survey recent work that uses randomization and additional information to beat these bounds. First we summarize [2] and [5] who show an optimal randomized strategy for the special class of node-disjoint $s$-$t$ path graphs. Next we look at [3]'s work on randomized algorithms in more general graphs. Finally, we look at recent work [1] on "learning" algorithms, which have a source of information of unknown quality and must use this to arrive at a solution somewhere between the online and offline optima. We conclude by posing a novel variant of $k$-CTP, which we analyze for some simple cases.

## 1 Introduction

The ***Canadian Traveller problem*** (CTP) was first introduced by Papadimitriou and Yannakakis[4] in 1991. In the CTP a traveller starting at vertex $s$ aims to move along the graph to $t$ in the shortest distance, where the traveller has access to the graph but does not know which edges are blocked. Upon arriving at a vertex the traveler discovers whether any edges leaving the vertex are blocked. An algorithm for solving CTP's competitiveness is measured relative to an adversary who knows which edges are blocked and can therefore take the shortest unblocked path. (Note that this is only meaningful when there exists at least one unblocked $s$-$t$ path, so we will assume this fact.)

Papadimitriou and Yannakakis showed that CTP is in general not tractable, i.e. achieving a bounded competitive ratio is PSPACE-complete. Therefore most work focuses on finding competitive strategies for ***k-CTP***, where there are at most $k=O(1)$ blocked edges and $k$ is known to the algorithm.

CTP has found applications in many real world situations. For example, navigating a road networks during severe weather conditions which may lead to e.g. roads being snowed out, or routing packets through a communication network with potential outages. In communication networks $k$ corresponds to the number of connections that the system can tolerate being down without error; in this situation it is quite reasonable that packets may have to navigate the network without perfect information.

### 1.1 Lower Bounds

A natural strategy is GREEDY: at any point in time, take the path that looks shortest from your current location to the destination. Somewhat surprisingly Xu et al. [7], show there are graphs where GREEDY performs very poorly, achieving a ratio of $\Omega(2^k)$ (see Figure 1). Further analysis in the paper shows GREEDY performs well with a $O(1)$ competitive ratio on grid-like graphs, which might explain why it feels intuitive.

Westphal [6] presents two lower bounds on algorithms for $k$-CTP. The graph in Figure 2 has $k+1$ equal-cost node-disjoint paths, all but one of which are blocked. Given a deterministic algorithm, an adversary could always make the unblocked path chosen last and block the end of every other path, demonstrating a $2k+1$ lower bound on the competitive ratio. Matching this $2k+1$ lower bound is achievable on any graph using the simple BACKTRACK strategy: start by trying the shortest path. Upon encountering a blocked edge, return to the start and try the next shortest path. This accumulates a distance at most $2k+1$ times the optimal distance since one only backtracks a maximum of $k$ times before finding the optimal path. For randomized algorithms, competitive ratio is measured as expected value of distance

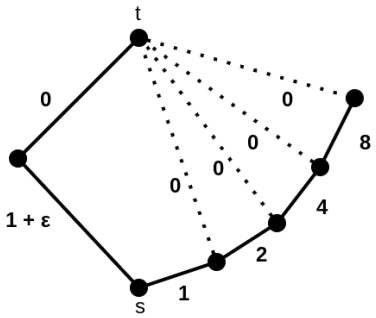


Figure 1: A graph where GREEDY achieves ratio $\Omega(2^k)$ – it always thinks that it can do better by stepping right instead of backtracking, but just keeps digging itself deeper into a hole.

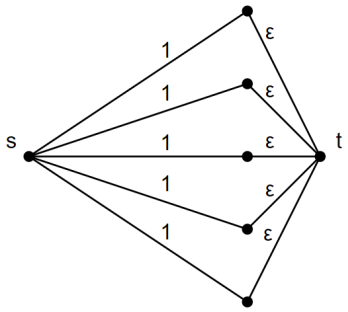


Figure 2: Lower bound: $2k+1$ worst case, $k+1$ average case

travelled divided by the optimum. Yao's principle can be applied to Figure 2 to show that if the input path is randomly selected, the competitive ratio of the deterministic algorithm is at least $k+1$. Therefore, the ratio achievable by randomized algorithm is lower bounded by $k+1$.

### 1.2 Outline

In Section 2 we summarize Bender and Westphal [2]'s randomized strategy that achieves competitive ratio of $k+1$ in the special case where all $s$-$t$ paths are node-disjoint. We discuss Shiri and Salman [5]'s correction to Bender and Westphal's algorithm. In Section 3 we present Demaine et al. [3]'s methods for achieving a ratio of $(1+1/\sqrt{2})k+1$ in pseudopolynomial time on general graphs. In Section 4 we summarize Bampis et al.'s [1] work on strategies for "learning" algorithms which are given predictions of the blocked paths. In Section 5 we present a novel variant of the $k$-CTP and analyze it for simple families of graphs.

## 2 Bender and Westphal

In [2] Bender and Westphal analyze a restricted version of the $k$-CTP problems where all $s$-$t$ paths are node-disjoint (i.e. share no vertices besides $s$ and $t$). Our analysis parallels their work, but places emphasis on making the intuition transparent. We also simplify their math, providing a clearer view for why this strategy works. We synthesize their ideas with Shiri and Salman's work [5], which points out a critical flaw in the analysis presented in [2].

Graphs with node-disjoint paths are a natural simplification of $k$-CTP: for such graphs, every edge blockage eliminates only a single path. So, we can view the problem as presenting $m$ disjoint $s$-$t$ paths, and allowing at most $k$ of them to be blocked. Note that if $k+1<m$, one of the shortest $k+1$ paths in the graph must be unblocked – so, we can without loss of generality ignore all but the $k+1$ shortest paths and assume $m \leq k+1$.

Any reasonable randomized strategy for this problem is of the following form: "Choose a probability distribution over the $k+1$ paths (based on their costs). Pick a path according to the assigned probabilities and follow it. If it is blocked return to the start. Remove the blocked path from consideration, recalculate the probability distribution, and repeat until the end is reached." We will show that for appropriate choice of probability distribution we can achieve a competitive ratio of $k+1$, which is tight in general.

### 2.1 A Simple Case: $k=1$

To develop intuition we consider the case $k=1$ (that is, the case in which at most one edge may be blocked). Recall that this allows us to without loss of generality consider only graphs with exactly 2 $s$-$t$ paths. Let the initial probability distribution over the paths be $p_1$,$p_2$ and let the path costs be $c_1$,$c_2$.

In the worst case we will spend the entire cost of a path before learning that it is blocked. If the first path is blocked, the competitive ratio is

$$\frac{p_1(2c_1+c_2)+p_2(c_2)}{c_2} = 2p_1 \cdot c_1/c_2 + 1,$$

because we pay $2c_1$ for back-tracking in the event that we start by guessing path 1. Symmetrically, if the second path is blocked the competitive ratio is $2p_2 \cdot c_2/c_1 + 1$. If none of the paths are blocked off, then the situation is strictly better for the competitive ratio, so this case can be safely ignored. The competitive ratio of the algorithm will be the max of the ratio achieved in the two situations. Balancing to optimize our ratio gives $p_1 \cdot c_1^2 = p_2 \cdot c_2^2$ which combined with $p_1+p_2=1$ implies:

$$p_1 = \frac{c_2^2}{c_1^2+c_2^2} \qquad p_2 = \frac{c_1^2}{c_1^2+c_2^2}$$

This captures our basic intuition: the larger $c_1$ is, the more likely we should be to choose the second path, and vice versa. For this choice of $p_1$,$p_2$ our ratio is:

$$\frac{(c_1+c_2)^2}{c_1^2+c_2^2} \leq 2 = k+1$$

## 2.2 Similar Costs Case

Now we attempt to generalize this computation to $k>1$ in the obvious manner: through induction on $k$. Our approach will require a "similar cost" condition, which will come out of the analysis. Later, we show how to eliminate the need for the similar cost condition.

Let the path costs be $c_1$,$c_2$,...,$c_{k+1}$; we seek to assign probabilities be $p_1, p_2, \ldots, p_{k+1}$ to the paths. If the optimal path has cost $c_j$, our competitive ratio is at most (by induction):

$$\frac{p_j c_j + \sum_{i \neq j} p_i (2c_i + k c_j)}{c_j}$$
$$= \sum_{i \neq j} \left( \frac{2c_i}{c_j} \right) p_i - (k-1) p_j + k.$$

Now, we balance the $k+1$ equations to optimize our competitive ratio. Let the common value that all of the equations are equal to be $d$. Then, the balancing can be expressed as the following matrix equation:

$$\begin{pmatrix} 1-k & 2c_2/c_1 & \ldots & 2c_{k+1}/c_1 \\ 2c_1/c_2 & 1-k & \ldots & 2c_{k+1}/c_2 \\ \vdots & \vdots & \ddots & \vdots \\ 2c_1/c_{k+1} & 2c_2/c_{k+1} & \ldots & 1-k \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_{k+1} \end{pmatrix} = \begin{pmatrix} d \\ d \\ \vdots \\ d \end{pmatrix}$$

Taking the inverse of the left matrix using computer software and solving for $p_i$ gives the following expression:

$$p_i = \frac{2\sum_{i \neq j} c_j + (1-k)c_i}{(k+1)^2 c_i} d.$$

$d$ can then be scaled to ensure that $\sum p_i = 1$. The resulting probability distribution must then be a multiple of the following distribution $\overline{p_i}$:

$$\overline{p_i} = 2\sum_{j \neq i} \frac{c_j}{c_i} + (1-k) = 2\sum_{j=1}^{k+1} \frac{c_j}{c_i} - (k+1)$$

For this to lead to a valid probability distribution, we need $\forall i : \overline{p_i} \geq 0$. Otherwise, some probabilities will be negative and we will not be able to follow this probability distribution. Rearranging this with simple algebra, we obtain that this probability distribution is possible when the ***similar costs property*** is satisfied.

**Definition 1** (Similar Costs Property)**.** $t$ paths with costs $c_1$,$c_2$,...,$c_t$ satisfy the ***similar costs property*** if for all $i=1,...,t$ it holds that

$$c_i \leq \frac{2}{t} \sum_{j=1}^{t} c_j$$

However, the similar costs property as presented in [2] is not sufficient because after finding a blocked path, eliminating it from consideration, and then recalculating the probability distribution, the remaining distribution must also satisfy the similar costs property. As such, we define the ***strong similar costs property*** as introduced by Shiri and Salman[5] in their correction:

**Definition 2** (Strong Similar Costs Property)**.** The set of paths $P$ satisfy the ***strong similar costs property*** if for all nonempty subsets $q \subseteq P$, $q$ satisfies the similar costs property.

By the above analysis, we present the following algorithm for node-disjoint-paths graphs satisfying the strong similar costs property:

**Algorithm 1** SRA (Simple Randomized Algorithm)

```
P ← {}
Add the shortest k+1 paths to P
while not reached destination do
    Run SRS on P
```

**Algorithm 2** SRS (Simple Randomized Step)

```
t ← number of paths left                ▷ functions like k+1, but updates as we find blockages
p_i ← 2 Σ_{j=1}^{t} c_j/c_i − t for i=1,...,|P|
s ← Σ p_i
p_i ← p_i / s
Select a random path P* where the probability of choosing the path with cost c_i is p_i
Traverse down P*
if a blocked edge is reached then
    Return to start
    P ← P \ P*
```

**Theorem 1.** When the strong similar costs property is satisfied, SRA is $k+1$ competitive.

*Proof.* In the above calculations, note that $d$ represents the value that all expressions are equal to and that $k+d$ represents the competitive ratio. All of the probabilities must sum to 1, so solving for $d$ yields:

$$\frac{d}{(k+1)^2} \sum_{i=1}^{k+1} \frac{2\left(\sum_{j \neq i} c_j\right) - (k+1)c_i}{c_i} = 1$$

$$d=(k+1)^2/\left(\left(\sum_{i=1}^{k+1}\sum_{j\neq i}c_j/c_i\right)-(k+1)(k-1)\right)$$

To bound this, we apply the AM-GM inequality, which tells us that the arithmetic mean of a list of nonnegative numbers is lower bounded by their geometric mean. Multiplying together all of the terms in the summation in the denominator yields 1, so the summation in the denominator is lower bounded by the number of terms, or $k(k+1)$. This yields the following upper bound for $d$:

$$d\leq\frac{(k+1)^2}{2k(k+1)-(k-1)(k+1)}=\frac{(k+1)^2}{k^2+2k+1}=1$$

Therefore, the competitive ratio is $k+d\leq k+1$ as desired. □

In Bender and Westphal's original paper, they assumed that the similar costs property implies the strong similar costs property. This assumption was countered in [5], which presents the costs $c_1=1, c_2=1, c_3=4, c_4=6$. The resulting probability distribution is $p_1=15/32, p_2=15/32, p_3=2/32, p_4=0$. If the path with cost $c_3=4$ is taken and is revealed to be blocked, then the remaining path costs do not satisfy the similar costs property as $6>\frac{2}{3}(1+1+6)$. As a result, Bender and Westphal's further analysis is flawed. The this section will be presented in accordance to Shiri and Salman's work, which offers a correction.

## 2.3 General Case with Partitioning

We've shown a $k+1$ competitive algorithms for graphs with the strong similar costs property, but this property does not hold in general. To handle the general case, we introduce the idea of "partitioning." When the remaining paths do not follow the similar costs property, we first sort them by cost. We then partition them into "classes" that each have the similar costs property. To do this partitioning, we greedily adding paths into the largest class, creating a new class only when adding the next edge would break the similar costs property.

We then apply our simple random step algorithm to these classes that satisfy the similar costs property until the destination is reached. This algorithm, developed by Shiri and Salman, is denoted as M-RBS (modified randomized backtrack strategy).

**Algorithm 3** M-RBS

```
S ← Partition(k+1 shortest paths)
while not reached destination do
    Run SRS on S[0]
    if S[0] still satisfies similar cost property and is nonempty then
        Continue
    P ← S[0] and remove S[0] from S
    Add Partition(P) to the beginning of S
```

**Algorithm 4** Partition

```
S ← [] and C ← {}
for path P_i in P in decreasing order of cost do
    if P_i ∪ C_i satisfies similar cost property then
        Add P_i to C
    else
        Add C to front of S and C ← {}
Add C to front of S if C ≠ {}
return S
```

**Lemma 1.** When tun on $t$ paths that satisfy the similar cost property such that at least one is traversable, M-RBS yields a $t$ competitive ratio.

*Proof.* For the base case of $t=2$, all non-empty subsets satisfy the similar cost property. Then M-RBS runs the same as SRA, yielding a competitive ratio of 2.

We proceed by induction on $t$ and assume that the claim is true for $t'<t$. We first establish some definitions. Let the paths be labeled $P_i\in P$ and let the probability distribution defined over them be $\Omega=(p_1,p_2,...,p_t)$. Let $P_i^*$ be the optimal path, with cost $c_i^*$ and probability of being taken be $p_i^*$. Finally, let $B$ be the set of blocked paths, $T$ be the set traversable paths, and $P_j$ be the path chosen by the algorithm. The competitive ratio $r$ is then:

$$r=\sum_{i\in T}\frac{p_ic_i}{c_i^*}+\sum_{j\in B}p_j\frac{2c_j+C^{t-1}}{c_i^*}$$

$C^{t-1}$ denotes the cost of the strategy after taking the randomly chosen path, and we claim that $C_{t-1}\leq(t-1)c_i^*$. If $P\backslash P_j$ satisfies the similar cost property, then $C_{t-1}\leq(t-1)c_i^*$ via induction.

For the case where the remaining paths do not fulfil the similar costs property, the paths are partitioned into $L$ classes $C_1,C_2,...,C_L$. Let $n_l$ be the size of class $C_l$, $\bar{c}_l$ be the sum of costs of paths in $C_l$, and suppose that $C_{l^*}$ is the class that contains $P_i^*$. Classes $C_1,C_2,...,C_{l^*-1}$ contain all blocked paths, causing the algorithm to incur cost $\sum_{l=1}^{l^*-1}2\bar{c}_l$. Class $C_{l^*}$ is a set of paths that has at least one traversable path and satisfies the similar cost

property, meaning our inductive assumption applies to it for $t'=n^{l^*}$. $C^{t-1}$ is then:

$$\sum_{l=1}^{l^*-1} 2\overline{c_l}+n_{l^*}\cdot c_{i^*}$$

Note that since $c_i^*$ is in a separate class from all classes $C_l$ such that $l=1,...,l^*-1$, $c_i^*$ does not satisfy the similar cost property when added to them. This implies a bound for $c_i^*$ which can be substituted into the expression for $C^{t-1}$:

$$c_i^* > \frac{2(\overline{c}_l+c_{i^*})}{n_l+1} \Rightarrow 2\overline{c}_l < c_i^*(n_l-1)$$

$$C^{t-1} < \sum_{l=1}^{l^*-1}(n_l-1)c_{i^*}+n_{l^*}\cdot c_{i^*} \le \sum_{l=1}^{l^*} n_l\cdot c_{i^*} \le (t-1)c_{i^*}$$

Therefore, our claim upper bounding $C^{t-1}$ is proven. Substituting this in and combining the summations for the competitive ratio $r$ yields:

$$r \le p_{i^*} + \sum_{j=1|j\neq i}^{t} p_j\left(\frac{2c_j}{c_{i^*}}+t-1\right)$$

To upper bound this with $t$ as desired, the following inequality needs to hold $\forall i=1,2,...,t$:

$$(2-t)p_i + \sum_{j=1|j\neq i}^{t} 2\frac{c_j}{c_i}p_j \le 1$$

To show this, we look back at how $p_i$ was defined. In the beginning of Section 2.2, we had the following expression which we later proved was at most $k+1$ if we chose $p_i$ in the way we did.

$$\alpha_k = \sum_{i=1|i\neq j}^{k+1} p_i\frac{2c_i}{c_j} - p_j(k-1)+k \le k+1$$

We make the substitution $k+1=t$ since they both represent the number of paths to obtain the desired inequality, thus proving that the algorithm is $t$ competitive. □

**Theorem 2.** M-RBS is $k+1$ competitive on graphs where all $s-t$ paths are node-disjoint

*Proof.* If the $k+1$ shortest paths already satisfy the similar costs property, then Lemma 1 directly gives the $k+1$ competitive ratio. If not, then the same logic in the proof of Lemma 1 used to upper bound $C^{t-1}$ can be applied to upper bound the competitive ratio of M-RBS as $tc_{i^*}$. $t=k+1$ in this case, giving a $k+1$ competitive ratio even when partitioning is required. □

**Theorem 3.** M-RBS runs in $O(t^2)$ time where $t=\min(n,k+1)$ (excluding the time it takes to traverse the graph). This bound is tight.

*Proof.* By restricting our focus to the $k+1$ shortest paths, we only work with $t$ paths. Sorting the paths takes $O(t\log t)$ time, and the outer while loop can run $t$ times. Partition runs in $O(t)$ time with the paths already sorted, and SRS takes $O(t)$ time. Therefore, the algorithm's runtime can be upper bounded as $O(t^2)$.

The bound is tight under the following circumstances: there is initially one class of size $t$. After each iteration, a path is removed, breaking the similar cost property. The class is then partitioned into one of size $t-2$ and one of size 1. This repeats, causing a partition to be ran each iteration leading to a $O(t^2)$ runtime.

To construct such an example, we start with the base (1,1,4). We then demonstrate how to take a class with $t$ paths and add another element $c_{t+1}$ such all paths initially follow the similar costs property. However, if the path with cost $c_t$ is chosen, then $c_{t+1}$ will be placed into a separate class of size 1. This leads to the following inequality:

$$\frac{2}{t}\sum_{i=1}^{t-1} c_i < c_{t+1} \le \frac{2}{t+1}\sum_{i=1}^{t} c_i$$

We claim that choosing $c_{t+1}$ to match the upper bound will also satisfy the lower bound. To show this we need to show that the lower bound is less than the upper bound. This inequality can be rearranged as:

$$\frac{2}{t}\sum_{i=1}^{t-1} c_i \le \frac{2}{t+1}\sum_{i=1}^{t} c_i \Rightarrow c_t \ge \frac{1}{t}\sum_{i=1}^{t-1} c_i$$

We claim that this inequality is always satisfied. We do so by induction on $t$. Our base case of (1,1,4) already satisfies this property. For larger sizes, the element $c_t$ was added such that it equals the upper bound above:

$$c_t = \frac{2}{t}\sum_{i=1}^{t-1} c_i \ge \frac{1}{t}\sum_{i=1}^{t-1} c_i$$

This completes the induction, showing that we can always add $c_{t+1}=\frac{2}{t+1}\sum_{i=1}^{t} c_i$ to generate an input that causes a $O(t^2)$ runtime. □

# 3 Demaine et al.

Westphal's analysis only applies to node-disjoint paths, and it is quite tricky to generalize. A large part of the problem for more general graphs is the fact that there may be exponentially many short $s$-$t$ paths, and that now there are dependencies between the paths.

Demaine et al. give a truly remarkable algorithm for beating the $2k+1$ deterministic lower bound on deterministic algorithms in the randomized setting for

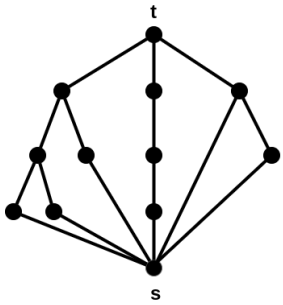


Figure 3: An example of an apex tree – with removal of $s$, this graph is a tree rooted at $t$.

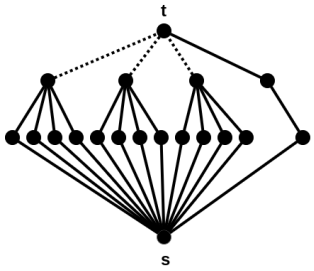


Figure 4: An apex tree in which a uniform bottom-up traversal performs poorly. Although there are at most $k$ blockages, because there is branching involved far more than a $\frac{k}{k+1}$ fraction of edges out of $s$ lead to blocked paths.

general graphs. Demaine et al.'s methods can be used to achieve a ratio of $(1+1/\sqrt{2})k+1$ in the general setting in pseudo-polynomial time. To the best of our knowledge it remains open whether the lower bound of $k+1$ is achievable in general.

### 3.1 Apex Trees

First, Demaine et al. give an optimal randomized strategy for $k$-CTP on a specific class of graphs, which he calls ***apex trees*** with cost identical paths. An apex tree is not quite a tree, but on removing $s$ becomes a tree (Figure 3). On identical cost apex trees it is impossible to achieve a competitive ratio of better than $k+1$, and the ***TRAVERSE-TREE*** algorithm achieves this. This algorithm will serve as an important subroutine for Demaine et al.'s method of giving a randomized strategy for general graphs.

The Traverse Tree algorithm works as follows: Starting from $t$, imagine a walk down the apex tree to $s$, choosing edges to descend the apex tree uniformly at random from those available at each step. Try the given path. If it fails at some location delete all implicated blocked paths and then find the lowest potentially reachable ancestor of the fail location. Choose a random path down from this new target, again choosing the edges to add to the path uniformly at random at each step. Now we try this new path, and we repeat this process until finding an unblocked path.

Note that this is randomly selecting paths in a top-down approach (choosing among the edges incident to $t$ and recursing downwards) as opposed to the natural bottom-up approach of assigning equal probability to all edges out of $s$. To see why that strategy fails, consider a graph where there are many large subtrees out of $t$, all blocked at the highest edge, and then a single unblocked path, as in Figure 4. If we repeatedly choose paths out of $s$ uniformly, we're very unlikely to choose the unblocked path before exploring too many of the blocked ones.

However, assigning probabilities in a top down matter and then traversing in this "depth first search" manner serves to balance the probabilities appropriately.

**Lemma 2.** The TRAVERSE-TREE strategy achieves a competitive ratio of $k+1$ on equal path cost apex trees.

*Proof.* In the Demaine et al. paper they do some complicated algebra to prove this, but it is actually quite straightforward to see why TRAVERSE-TREE works.

Let the cost of all paths be 1. Without loss of generality no edges out of $s$ are cut (these are edges that can be seen from the start, so the adversary gains nothing from blocking these).

First, consider an apex tree where all paths are of height 1: this is simply a set of node-disjoint paths. Consider the chance that we decide to traverse a particular blocked path before the unblocked path. Clearly this is 1/2: either one is equally likely to be chosen. Thus, in this case the expected cost is at most $\frac{1}{2}2k+1=k+1$, as desired.

Now we consider a general example and argue by induction that the competitive ratio is still $k+1$. Let the number of blocked edges in the subtrees of $t$ be $b_1$,...,$b_\ell$. We are equally likely to visit a particular wrong subtree before or after the correct subtree. If we traverse a subtree $i$ that is completely blocked by the $b_i$ blockages, we may incur time up to $2b_i$ on the subtree before abandoning it. Let $i_*$ denote an unblocked subtree. Using induction we know that we spend at most $b_{i_*}+1$ on the correct subtree in expectation. Overall, our expected time is at most:

$$\sum_{i\neq i_*}\frac{1}{2}2b_i+b_{i_*}+1=\sum b_i+1=k+1.$$

Thus it is shown that on equal cost apex trees we incur cost $k+1$. □

The classic randomized lower bound (see introduction) is in fact an apex tree, so $k+1$ is tight.

### 3.2 Extending to general graphs

Now, we consider the question of computing a competitive strategy in a general graph. Surprisingly, by combining greedily choosing shortest paths with using TRAVERSE-TREE on a apex tree that arises from considering nearly shortest paths we can achieve better competitive ratios than are possible for deterministic algorithms. Demaine's algorithm, called the Greedy & Reposition Randomized (GRR) strategy, is summarized in the following pseudo code:

**Algorithm 5** GRR

1: $i \leftarrow 0$ counts the number of blockages that we have found thus far.
2: **while** not done **do**
3: $S \leftarrow$ paths of length at most $(1+\alpha)$ factor longer than the shortest path, given our current knowledge of blocked paths.
4: with probability $\frac{k-i}{k-i+1}$ (recall $k-i$ is the number of potential blocked edges that we have not found yet) uniformly randomly choose a shortest path to try.
5: with probability $\frac{1}{k-i+1}$ commit to a TRAVERSE-TREE on $S$. (It turns out that we can represent $S$ as an apex tree.)
6: When a blocked edge is encountered, return to the start and increment $i$.

Note that we will continue in a TRAVERSE-TREE operation until the entire tree being traversed is found to contain no $s$-$t$ paths. The details of how to select paths are largely addressed in subsection 3.3 which discusses Demaine's "implicit path representation" scheme.

Intuitively the algorithm works for the following reason: we have an optimal method for traversing equal cost apex trees, but need to be cautious about applying it to non equal cost apex trees. Thus, we balance cautiously doing apex tree traversal and simply greedily choosing shortest paths. The proof enhances this intuition.

**Theorem 4.** GRR is $(1+1/\sqrt{2})k+1$ competitive for $k$-CTP and can be computed in pseudo-polynomial time.

*Proof.* Let $c(i)$ be the amount of work done by TRAVERSE-TREE if it is called by the $i$-th iteration of the loop. Let $d_*$ represent the length of path that OPT takes.

**Case 1:** The traveller does not learn about all blocked edges. In this case she could have either ended on some TRAVERSE-TREE operation or by following a shortest path. In expectation the distance travelled is at most

$$\sum_{i=1}^{k} \frac{c(i)}{k+1} + \sum_{i=0}^{k-1} \frac{k-i}{k+1} 2d_*. \tag{1}$$

This is because there is a $\frac{k-i+1}{k+1}$ chance of the loop progressing to level $i$ and then a $\frac{1}{k-i+1}$ chance of choosing to do a TRAVERSE-TREE at that point. Note that failed TRAVERSE-TREE and failed greedy path choices alike both incur at most $2d_*$ cost. Thus Equation 1 accurately describes the cost. We can bound $c(i)$ on a successful TRAVERSE-TREE as $c(i) \leq (1+\alpha)(k-i+1)d_*$ because TRAVERSE-TREE is basically optimal (we lose the $1+\alpha$ factor because it's not actually equal cost paths, which was the assumption in our apex tree analysis). Using our bound on $c(i)$ and taking the sums, we find that in Case 1 our expected competitive ratio is bounded by

$$\frac{3+\alpha}{2}k+1.$$

**Case 2:** Another possibility is that the traveller learns all the blocked edges, at which point she can return to the start and take an optimal path. In this case, we cannot end in an TRAVERSE-TREE. However, we are still competitive because the failure of TRAVERSE-TREE implies that the actual shortest path is more than a factor of $(1+\alpha)$ larger than our guesses to the shortest path, so our previous steps were sufficiently cheap. Thus the travellers time in this case is at most

$$d_* + \sum_{i=1}^{k} \frac{c(i)}{k+1} + \sum_{i=0}^{k-1} \frac{k-i}{k+1} 2d_*/(1+\alpha).$$

Where now $c(i) \leq 2(k-i+1)$. Simplifying and dividing by $d_*$ to compute our competitive ratio we find the ratio achieved in Case 2 to be

$$\frac{2+\alpha}{1+\alpha}k+1.$$

To balance Case 1 and Case 2 we set $\alpha = \sqrt{2}-1$ and obtain a randomized algorithm with competitive ratio $(1+1/\sqrt{2})k+1$. The run time guarantee follows from analysis in Subsection 3.3. □

### 3.3 Implicit representation of all near shortest paths

It remains to see how we can efficiently implicitly represent shortest paths and "nearly shortest paths". We start by computing the $\mu$ shortest distances to each vertex with a procedure similar to Dijkstra's algorithm, as depicted in Algorithm 3.3. We remark that the description of Algorithm 3.3 has been oversimplified in our presentation here for ease of explanation. In reality you need to be more careful that the changes to $D(v)$ are happen in phases, i.e. the updates to $D(u)$ are performed only after looping through $u,v$ and determining what updates need to be performed, but this is a minor detail.

**Algorithm 6** $\mu$-shortest distances

$Q \leftarrow \varnothing$ stores a "frontier" of vertices with neighbors that need to be updated
Initialize $D(v)$, which will store the $\mu$ shortest distances $v \rightsquigarrow t$ found so far, to $\{\infty\}$ for $v \neq t$ and $\{0\}$ for $v = t$.
**for** $i \leftarrow 1,...,\mu|E|$ **do**
    $Q' \leftarrow \varnothing$
    **for** $u,v$ with $v \in Q$, $u \rightarrow v \in E$ **do**
        Update $D(u)$ based on $D(v)$ and $d(u,v)$.
        Push $u$ to $Q'$
    $Q \leftarrow Q'$

Now, we want an implicit representation of the paths. This is accomplished by just running Algorithm 3.3 in reverse! More precisely, now we start at $s$ and propagate forwards, finding paths that are $\mu$ shortest paths. Whenever we find an edge and a vertex that must be included in the graph of the $\mu$ shortest paths we record it. Through this method we create a new graph $G' = (V', E')$. In particular, as we propagate forward, we compute lists $L(v)$ of at most $\mu$ distances, corresponding to the possible $\mu$ shortest distance paths from $v$ to $t$ that have come from a $\mu$ shortest path out of $s$ indirectly. This graph may of course contain cycles. We would like to eliminate the cycles and reduce the structure so that we have an Apex tree. We can essentially accomplish this by creating a lot of copies of all the vertices.

For each $\ell \in L(v)$ we create a new vertex $v_\ell$. Now, we create edges $v_\ell \rightarrow w_{\ell'}$ when $(v,w) \in E'$ and $\ell - d(v,w) = \ell'$. This of course removes the cycles from the graph. Demaine et al. observe that this new graph can now represent an apex tree. Taking $\mu$ to be the sum of all edge weights suffices to represent all paths within a factor of $1+\alpha$ of the shortest path among those not yet discovered to be blocked. Because $\mu$ is pseudo-polynomial the algorithm's run is pseudo-polynomial, as claimed.

# 4 Bampis, Escoffier, Xefteris: Learning-Augmentation

We have seen that clever use of randomization can provide a substantial improvement in worst-case competitive ratio, reducing us to $\left(\frac{1}{\sqrt{2}}+1\right)k+1$ in general, or $(k+1)$ in graphs with node-disjoint paths. However, in many more realistic applications, we might find these algorithms unsatisfying. It is often the case that, although we don't know for *certain* which edges are blocked, we have reasonably strong confidence – as opposed to being truly adversarial, the problem may be related to some real-world data with patterns we can extract predictions from.

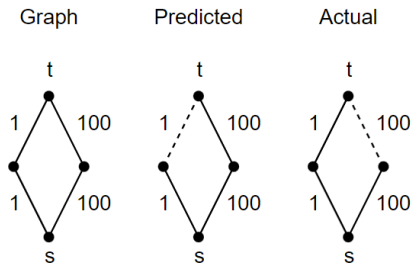


Figure 5: The optimal path has length 2, but trusting bad predictions gives us a path of length 202. This gives competitive ratio of 101 with $k = 1$.

For that sort of instance, backtracking algorithms seem incredibly wasteful! Why should we spend so much time exploring paths we are pretty sure are going to be blocked anyway when, if our predictions were correct, we could have just gone down the optimal path to begin with?

In a recent paper, Bampis, Escoffier and Xefteris consider how to resolve this problem. They use the concept of a *learning-augmented algorithm*: an algorithm that's provided with a prediction of which edges are blocked (e.g. by a machine learning model trained on previous instances of the problem), and seeks to perform close to optimally if the predictions are good.

## 4.1 Learning-Augmented Approach

Of course, if we are given known-correct predictions, it is easy to get optimal performance – just follow the shortest path in the predicted graph! The issue arises when we are unable to fully trust our predictions. Perhaps we are training an AI model that correctly predicts road blockages sometimes, but has the potential to be wildly wrong. In this case, total blind trust may lead us catastrophically astray, as shown in Figure 5.

Bampis, Escoffier and Xefteris quantify this degree of "trust" in our predictions with a parameter $\varepsilon$: if we want to ensure that we always get within $(1+\varepsilon)$ of optimal when the predictions were perfect, how far must we allow ourselves to be led astray when the predictions are wrong? We provide here a simplified presentation of the key results of the paper: proving and achieving lower-bounds on performance with incorrect predictions for deterministic algorithms, and improving on this performance with a randomized algorithm.

## 4.2 Deterministic Case

The deterministic algorithm Bampis et al. present for this problem follows a very intuitive strategy: since we don't have complete trust in our predictions, we first run

an incomplete version of BACKTRACK to ensure that our predictions haven't caused us to miss a much shorter path. Once we've spent some threshold amount of total work checking the very shortest paths, we then pause and try going down the optimal path in the predicted graph. If that predicted optimal path is blocked, we return to the start and resume BACKTRACK to completion.

This approach is formalized as follows:

**Algorithm 7** E-Backtrack

```
P_pred ← shortest path in the predicted graph
TVL ← 0 ▷ the total visited length by the algorithm
while t not reached do
    P_next ← shortest path not yet proven to be blocked (in the real graph)
    if P_pred is unblocked and w+2|P_next| > ε|P_pred| then
        Explore P_pred (returning to s if blocked)
    else
        Explore P_next (return to s if blocked)
        TVL ← TVL+length traveled in this step
```

This algorithm performs BACKTRACK until the moment BACKTRACK would cause it to traverse more than $\epsilon \cdot \text{OPT}_{pred}$ total path length, at which point it stops and attempts the predicted optimal path first. This clearly achieves a $(1+\varepsilon)$ competitive ratio under correct predictions, because it ensures we waste less than $\epsilon \cdot$OPT work before trying the optimal path. If we were given incorrect predictions, there are 2 cases:

**Case 1:** We may find $t$ before exploring $P_{pred}$. In this case, E-BACKTRACK behaves identically to BACKTRACK, so is $(2k+1)$-competitive

**Case 2:** We may explore $P_{pred}$ before finding $t$. This means that, before trying the optimal path, we try and backtrack on at most $k-2$ paths of lengths at most OPT, plus $P_{pred}$, making our total work $2(k-1)\cdot\text{OPT}+2P_{pred}+\text{OPT}$. But now, note that we only take $P_{pred}$ if the next backtrack path would put us over $\epsilon|P_{pred}|$ work. So, $P_{pred}$ can be at most $\frac{1}{\varepsilon}(2(k-1)\cdot\text{OPT}+2\cdot\text{OPT}) = \frac{2k}{\varepsilon}$. Thus, in this case, we are $\left(2k-1+\frac{4k}{\varepsilon}\right)$-competitive.

For any $\varepsilon \leq 2k$, $2k+1 \leq 2k-1+\frac{4k}{\varepsilon}$, so we have found an algorithm with competitive ratio less than $1+\varepsilon$ given correct predictions, and $2k-1+4k/\varepsilon$ given incorrect predictions, for any $\epsilon \in (0, 2k]$. Figure 6 demonstrates that it's impossible to do better:

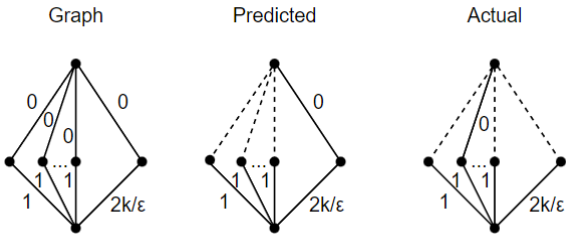


Figure 6: In this graph, to get a competitive ratio less than $(1+\varepsilon)$ if our predictions are correct, we must go down the $\frac{2k}{\varepsilon}$ edge first. If we decided to check all the cost-1 edges beforehand, and our predictions were correct, we'd end up with competitive ratio $\frac{2k+2k/\varepsilon}{2k/\varepsilon} = 1+\epsilon$. But, if our predictions were wrong, this strategy can result in us traversing all nonzero edges in the graph (except for one) twice, giving us a competitive ratio of $2(k-1)+2\left(\frac{2k}{\epsilon}\right)+1 = 2k-1+\frac{4k}{\epsilon}$.

## 4.3 Randomized Case

Drawing on our previous discussion of randomized k-CTP, one might now wonder whether it is possible to get results about randomized learning-augmented algorithms as well. Does a similar approach generalize?

Since tight bounds are not known for randomized k-CTP in general, Bampis et al. follow Bender and Westphal in focusing on the case of node-disjoint paths. We present the following modified form of E-BACKTRACK:

**Algorithm 8** E-RandBacktrack

```
P_pred ← shortest path in the predicted graph
TVL ← 0 ▷ the total visited length by the algorithm
while t not reached do
    P_next ← path selected by M-RBS on the graph excluding P_pred
    if P_pred is unblocked and w+2|P_next| > ε|P_pred|, or if P_pred is the only unblocked path then
        Explore P_pred (return to s if blocked)
    else
        Explore P_next (return to s if blocked)
        TVL ← TVL+length traveled in this step
```

Note that we ensure $(1+\varepsilon)$-competitiveness in the case of correct predictions by always exploring $P_{pred}$ before doing more than $\varepsilon|P_{pred}|$ work. Now, suppose our predictions were incorrect. Here, the expected work we do is bounded by the expected amount of work done by randomized backtracking, plus $2|P_{pred}|$ times the probability that we explore $P_{pred}$. Note that we are doing randomized backtracking on a problem with $k-1$ blocked edges,

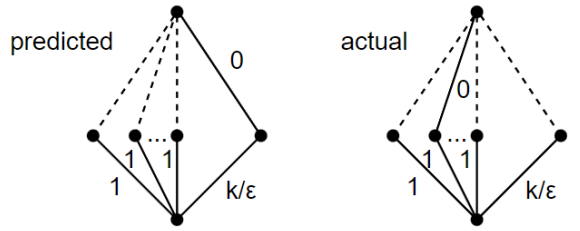


Figure 7: A graph similar to the one used to prove deterministic lower-bounds, but has a long path with cost $k/\varepsilon$. Note that any one of the short paths might be the unblocked one

so by Bender and Westphal's analysis, this requires at most $k \cdot OPT$ work. Now, note that we only explore $P_{pred}$ if $\text{TVL}+2|P_{next}| > \varepsilon|P_{pred}|$. If randomized backtracking has not yet terminated at this stage, this means randomized backtracking will do at least $\text{TVL}+|P_{next}|$ total work. So, we certainly never explore $P_{pred}$ unless the randomized backtracking part of our algorithm requires more than $\frac{\varepsilon|P_{pred}|}{2}$ work. Since we know the expected amount of work done by the randomized backtracking portion of our algorithm is bounded by $k$, we can apply the Markov bound to show that the probability it does more than $\frac{\varepsilon|P_{pred}|}{2}$ work is at most $\frac{2k}{\varepsilon|P_{pred}|}$. So, the expected amount of work we expend by exploring $P_{pred}$ is

$$Pr(\text{we explore } P_{pred}) \cdot 2|P_{pred}| \leq \frac{2k}{\varepsilon|P_{pred}|} \cdot 2|P_{pred}| = \frac{4k}{\varepsilon}$$

meaning that we get a competitive ratio in the incorrect-prediction case of at most $\left(k + \frac{4k}{\varepsilon}\right)$.

Unlike for deterministic algorithms though, even though we restricted to node-disjoint paths, we cannot present a tight lower-bound here. Bampis et al. speculate that this algorithm is not optimal, and that one exists that achieves $k + \frac{k}{\varepsilon}$. They present a proof that this is a lower bound as follows:

Suppose we are given a graph as above. Any deterministic algorithm for this problem is of the form "try $i$ of the short paths, then try the long path, then try the rest of the short paths". So, a randomized algorithm must be some probability distribution over these deterministic strategies. Let $p_i$ denote the probability that we select a strategy that tries $i$ short paths before trying the long path. If our predictions were correct, the long path is the optimal one, and every short path we try incurs a cost of 2. Thus, the expected cost is

$$\frac{k}{\varepsilon} + \sum_{i=0}^{k} 2i \cdot p_i$$

giving us competitive ratio

$$1 + \frac{\varepsilon \sum 2i \cdot p_i}{k}$$

So, in order for our randomized algorithm to be less than $(1+\varepsilon)$-competitive in expectation, it must be the case that

$$1 + \frac{\epsilon \sum 2i \cdot p_i}{k} < 1 + \varepsilon \;\Rightarrow\; \sum_{i=0}^{k} i \cdot p_i < \frac{k}{2}$$

Now, suppose our predictions were incorrect, as in Figure 7. That is to say, suppose instead of the long path being unblocked and all short paths blocked as predicted, the long path is blocked and some short path is blocked (chosen uniformly at random). Now, for a deterministic strategy, the expected distance is equal to the expected distance on the short paths plus the expected distance on the long path. If we ignore the long path, we observe that any deterministic strategy is equivalent to randomized backtracking, and so has expected distance $k$. Now, consider the algorithm that explores the long path after $i$ short paths. The probability that this algorithm has to explore the long path is equal to the chance that the unblocked path is not among the first $i$, so $1 - \frac{i}{k}$. Thus, the expected distance the algorithm travels on the long path is

$$2|P_{long}|\left(1 - \frac{i}{k}\right) = \frac{2k}{\varepsilon}\left(1 - \frac{i}{k}\right) = \frac{2k}{\varepsilon} - \frac{2i}{\varepsilon}$$

Recall that a randomized strategy is equivalent to some probability distribution of these deterministic strategies, so will have competitive ratio

$$k + \sum_{i=0}^{k} p_i \left(\frac{2k}{\varepsilon} - \frac{2i}{\varepsilon}\right) = k + \frac{2k}{\varepsilon} - \frac{2}{\varepsilon}\sum_{i=0}^{k} p_i \cdot i$$

Recalling that $\sum_{i=0}^{k} i \cdot p_i < \frac{k}{2}$, this competitive ratio must be greater than

$$k + \frac{2k}{\varepsilon} - \frac{2}{\varepsilon}\left(\frac{k}{2}\right) = k + \frac{k}{\varepsilon}$$

As mentioned, Bampis et al. believe this lower bound to be tight, but have been unable to produce an algorithm achieving it, even in the case of node-disjoint paths. A potential direction for further investigation would be to apply ideas from Demaine et al. to find randomized learning-augmented algorithms for general graphs.

# 5 Friendly Canadian Locals

We now briefly present a new class of related problems, offering proofs of basic facts while leaving most of the

serious questions open. As with the previous section, we'll consider a $k$-CTP variant in which the traveler is provided with additional information. However, in our variant the information is entirely trustworthy – it's just limited in size.

## 5.1 Problem Description

Suppose you were planning to travel through some Canadian town. When you get there, if there was a storm, you will not know which paths are snowed in, but your friend (who lives there) will. If your friend can only communicate a small amount of information once you get there (say, if the blizzard is so intense that the only way she can communicate with you is by a signal flare), how can you best agree on an encoding procedure beforehand so that this information is as useful as possible? More formally, the situation proceeds as follows:

1. First, an adversary gives us a graph $G$, a start node $s$, a destination node $t$, and a cost function $c$ from the edges of $G$ to positive real number travel costs. Based on this graph, the traveler and the local work together to develop a shared strategy of information encoding using $w$ bits.

2. Then, the adversary, aware of this strategy, chooses to block up to $k$ of the edges in the graph. The blocked edges are revealed to the local but not the traveler.

3. Based on the blocked edges, the local uses the shared encoding scheme to send $w$ bits of information to the traveler.

4. The traveler executes some algorithm informed by these $w$ bits to travel from $s$ to $t$ with as good a competitive ratio as possible against the true shortest path.

**Observation:** One important observation to make is that, if $w \geq k \cdot \log|E|$, where $|E|$ is the number of edges in $G$, it is possible for the local to perfectly encode the locations of every blocked edge in the graph – so, the traveler can obtain a competitive ratio of 1. We also know that if $w=0$ any deterministic algorithm will have a $2k+1$ competitive ratio lower-bound. However, it is not obvious what competitive ratios are achievable for $0<w<k \cdot \log|E|$.

## 5.2 Node-Disjoint Path Graphs

Although we will not present algorithms for Friendly Canadian Local problems on general graphs, we will consider as in Westphal et al. and Bampis et al. the special case of node-disjoint path graphs, as the problem is substantially simplified for such graphs. Given a graph with node-disjoint paths, and an information bound of $w$ bits, we propose an encoding scheme as follows:

**Simple Node-Disjoint Path Encoding Scheme:** Consider the $k+1$ shortest $s$-$t$ paths. The local and traveler agree in advance on a partition of these paths into $2^w$ arbitrary groups of $\frac{k+1}{2^w}$ paths each. Then, when the local sees the blockages, she uses her $w$ bits to inform the traveler of the index of the group containing the smallest unblocked path.

**Lemma 3.** This scheme allows the traveler to achieve a competitive ratio of $\frac{2k}{2^w}+1$ with a deterministic traversal strategy, or $\frac{k}{2^w}+1$ with a randomized strategy.

*Proof.* If the traveler restricts their attention to the specified group, they now have an instance of $k-$CTP with the same value of $OPT$ but at most $\frac{k}{2^2}$ blocked edges. So, using BACKTRACK or M-RBS on this group will achieve competitive ratios of $\frac{2k}{2^w}+1$ or $\frac{k}{2^w}+1$, respectively. □

This algorithm is straightforward and effective, but it lacks a potentially desirable property. Consider the graph in Figure 8:

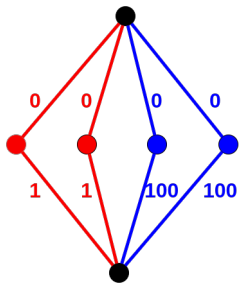


Figure 8: An instance of the 3-CTP where with no information, BACKTRACK obtains a ratio of 3.04. When given information from the friendly Canadian with $w=1$, we partition into the red and blue groups, but this only improves the competitive ratio to 3.

Although our partition scheme guarantees a $\frac{2k}{2^w}+1$ competitive ratio, it does not in general achieve better than this on graphs where a competitive ratio less than $2k+1$ was possible with no information. So, we propose the following modification that partitions dissimilar edge weights together to get the best results from our previous algorithms:

**Algorithm 9** SPP (Stratified Path Partitioning)

```
Initialize 2^w empty groups S_0, ..., S_{2^w-1}
for 0 ≤ j < k+1 do
    d ← j mod 2^w
    Add the jth shortest path to S_d
```

**Lemma 4.** If it is possible to achieve a competitive ratio of $r$ on a graph with node-disjoint paths, applying SPP achieves competitive ratio at most $\frac{r-1}{2^w}+1$ in the corresponding Friendly Canadian Local problem.

*Proof.* Note that no deterministic algorithm on node-disjoint paths should try a longer path before a shorter path, because an adversary would then do better by blocking the longer one. So, if it is possible to achieve competitive ratio $r$, BACKTRACK must achieve it. Consider the worst-case competitive ratio of an SPP-informed algorithm. Note that our SPP-informed algorithm performs exactly every $2^w$th step a full BACKTRACK would perform, because the groups are stratified across costs. For all paths that the SPP-informed algorithm backtracks on, full BACKTRACK must backtrack on paths at least as long $2^w$ times. BACKTRACK spends $(r-1)\cdot OPT$ distance backtracking, so all but the last path explored by the SPP algorithm can be explored with distance $\frac{(r-1)\cdot OPT}{2^w}$. This means it achieves an overall competitive ratio at most $\frac{(r-1)}{2^w}+1$. □

### 5.3 Further Questions

As with randomized algorithms, the node-disjoint paths case is relatively easy, but we expect to have to work harder to understand the case in general graphs. We know we can achieve a deterministic competitive ratio of $2\left(k-\frac{w}{\log_2|E|}\right)+1$ for $w \le k\cdot\log_2|E|$ by revealing a subset of the blocked edges. But can we do better in general? Are there special cases somewhat more general than node-disjoint-paths (e.g. apex trees) where we can find better encoding schemes?

Additionally, are there other variants of this problem with interesting properties? We considered changing the problem by having $s$,$t$ chosen by the adversary *after* the local sends the $w$ bits – this would require the local to be conveying information important to the graph as a whole, as opposed to just the specific journey the traveler wants to take. We also considered encoding schemes to be developed using randomness unknown to the adversary – do such schemes offer advantages? We hope that further investigation into these stories can provide interesting insights about what makes the CTP difficult, and what information is necessary in practice to do well.

## 6 Conclusion

We have surveyed various results about the Canadian Traveller's Problem, presenting several ways of breaking the $2k+1$ barrier on competitive ratio for deterministic algorithms. First, Bender and Westphal demonstrated that randomized algorithms could improve performance on the special case of graphs with node-disjoint paths. Then, Demaine et al. extended those results to show that randomized algorithms can achieve an improvement in competitive ratio over deterministic ones even on general graphs. Next, Bampis et al. presented a model in which a good competitive ratio can be achieved from a correct prediction of the graph, while still maintaining robustness if the prediction is wrong. Finally, we considered to what degree limited advance information is helpful in improving performance. Future work in this area to bring down the competitive ratio on general graphs to match the $k+1$ barrier or further analyzing the class of problems presented in this paper could prove valuable for tackling the real world problems (like traffic or communication networks) that can be modeled by $k$-CTP.

## References

[1] Evripidis Bampis, Bruno Escoffier, and Michalis Xefteris. Canadian traveller problem with predictions. In *Approximation and Online Algorithms: 20th International Workshop, WAOA 2022, Potsdam, Germany, September 8–9, 2022, Proceedings*, page 116–133, Berlin, Heidelberg, 2022. Springer-Verlag.

[2] Marco Bender and Stephan Westphal. An optimal randomized online algorithm for the k-canadian traveller problem on node-disjoint paths. *J. Comb. Optim.*, 30(1):87–96, jul 2015.

[3] Erik D. Demaine, Yamming Huang, Chung-Shou Liao, and Kunihiko Sadakane. Canadians should travel randomly. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming*, pages 380–391, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[4] Christos H. Papadimitriou and Mihalis Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84(1):127–150, 1991.

[5] Davood Shiri and F. Sibel Salman. On the randomized online strategies for the k-canadian traveler problem. *J. Comb. Optim.*, 38(1):254–267, jul 2019.

[6] Stephan Westphal. A note on the k-canadian traveller problem. *Information Processing Letters*, 106(3):87–89, 2008.

[7] Yinfeng Xu, Maolin Hu, Bing Su, Binhai Zhu, and Zhijun Zhu. The canadian traveller problem and its competitive analysis. *Journal of Combinatorial Optimization*, 18(2):195–205, Aug 2009.