Tight Bounds for the Advice Complexity of the Online Minimum Steiner Tree Problem*

Kfir Barhum

Department of Computer Science, ETH Zurich, Switzerland
barhumk@inf.ethz.ch

Abstract. In this work, we study the advice complexity of the online minimum Steiner tree problem (ST). Given a (known) graph $G = (V, E)$ endowed with a weight function on the edges, a set of N terminals are revealed in a step-wise manner. The algorithm maintains a sub-graph of chosen edges, and at each stage, chooses more edges from G to its solution such that the terminals revealed so far are connected in it. In the standard online setting this problem was studied and a tight bound of $O(\log(N))$ on its competitive ratio is known. Here, we study the power of non-uniform advice and fully characterize it. As a first result we show that using $q \cdot \log(|V|)$ advice bits, where $0 \leq q \leq N - 1$, it is possible to obtain an algorithm with a competitive ratio of $O(\log(N/q)$. We then show a matching lower bound for all values of q, and thus settle the question.

Keywords: Online algorithms, advice complexity, minimum Steiner tree.

1 Introduction

Online algorithms are a realistic model for making decisions under uncertainty. As opposed to classical computational problems, in the online setting the full input to the problem is not known in advance, but is revealed in a step-wise manner, and after each step the algorithm has to commit to a part of its solution. The *competitive analysis* of an algorithm, as introduced first in [1], measures the worst-case performance of the algorithm compared with the optimal offline solution to the respective optimal offline (classical) computational problem. We refer to [2] for a detailed introduction and survey of many classical online problems.

In recent years, motivated, among others, by the fact that for some problems (e.g. Knapsack [3]) no deterministic algorithm can admit any competitive ratio, the natural question, "how much information about the future is needed in order to produce a competitive solution?", was posed by Dobrev et al. [4] and Böckenhauer et al. [5], and independently by Emek et al. [6]. In this work we use the framework of Hromkovic et al. [7], that unifies the models and allows posing the question in its full generality: "What is the exact power of advice bits for some specific online problem?".

* This work was partially supported by SNF grant 200021141089.

V. Geffert et al. (Eds.): SOFSEM 2014, LNCS 8327, pp. 77–88, 2014.

In online computation with advice, the algorithm's machine has access to a special infinite advice string ϕ, produced by an oracle that has access to the entire input. The general goal is to try to characterize the dependence of the achievable competitive ratio to the maximal number of advice bits read from the advice tape.

In this work, we focus on the advice complexity of the online version of the well-studied minimum Steiner tree problem. In the offline setting, an instance $\mathcal{I}$ to ST is a graph $G = (V, E)$ endowed with a weight function on the edges $w : E \to \mathbb{R}^+$ and a set $T \subseteq V$ of vertices called terminals. A subgraph σ of G is a solution to the instance if every pair of terminals is connected in it. The cost of a solution σ, denoted $\text{cost}(\sigma)$, is the sum of the weights of the edges in it, and a solution is optimal if there exists no other solution with smaller cost.

Following previous work of Imase and Waxman [8], we consider the following natural online version of the minimum Steiner tree problem. Given a (known) weighted graph G, the terminals appear in a step-wise manner, and the algorithm maintains a subset of the edges as its solution. Upon receiving a new terminal, the algorithm extends the current solution so that the new terminal is connected to the old ones. The entire graph is known in advance, and only the specific subset of terminal vertices (and an ordering on it) is part of the instance.

More formally, given a ground graph G with a weight function w, an instance to $\mathsf{ST}(G, w)$ is an ordered list of vertices called terminals $[v_1, v_2, \ldots, v_N]$, where $v_i \in V$. At time step i, the algorithm receives terminal v_i and extends its current solution by choosing additional edges from G. The augmented solution computed by the algorithm by the end of step i is a solution to the offline problem on G with $\{v_1, \ldots, v_i\}$. As in the offline case, the cost of the solution is the total weight of edges chosen by the algorithm. An instance for $\mathsf{ST}(G, w)$ with N vertices is encoded canonically as a binary string of length $N \cdot \lceil \log(|V|) \rceil$.

An online algorithm with advice for $\mathsf{ST}(G, w)$ is strictly c-competitive using b advice bits if, for every instance, there exists an advice string ϕ such that the total weight of edges chosen by the algorithm during its computation with ϕ is at most c times the weight of the edges of an optimal solution to the offline problem, and at most b bits are read from ϕ. In general, $c = c(\cdot)$ and $b = b(\cdot)$ are function of some parameter of the input, typically the input length.

1.1 Our Contribution

We obtain a complete and exact characterization of the power of advice bits for the online Steiner tree problem.

In Section 2, we first give a variant to the greedy algorithm of [9] (without advice), which is $O(\log(N))$-competitive on an input with N terminals, and then show that our modified algorithm, which we call terminal-greedy algorithm, is $O(\log(\frac{N}{q}))$-competitive, utilizing an advice of size $q \cdot \log(|V|)$. Informally, the advice we employ is a description of the q most expensive terminals. Namely, the q terminals for which the terminal-greedy algorithm added the largest total weight of edges during its execution without advice.

In Section 3, we complement our algorithm with a matching lower bound, for the full range of advice bits.

We revisit the construction of [8], that shows a matching lower bound of $\Omega(\log(N))$ for the competitive ratio in the standard online setting (without advice). Inspired by their construction, we introduce Diamond graphs and study their properties. The construction they use can be viewed as a degenerated diamond graph. Our analysis takes a new approach using probabilistic arguments and requires a more general class of graphs in order to handle algorithms that use advice.

For every q s.t. $0 \leq q \leq N-1$ and an online algorithm taking advice of size $q \cdot \log(|V|)$, we construct a *different* instance distribution on a suitable Diamond graph. We then employ the mechanism developed earlier in order to show that for this graph there exists an instance for which the algorithm is $\Omega(\log(\frac{N}{q}))$-competitive. Our lower bound here holds already for the unweighted case, where $w(e) = 1$ for every $e \in E$.

We observe (details are omitted in this extended abstract) that a partial result of a matching lower bound for some values of advice size $q \log(|V|)$ can be obtained using the original construction presented in [8], albeit using a different analysis. We emphasize that our new construction is essential for the proof of a matching lower bound for the full range $0 \leq q \leq N-1$ of online algorithms using $q \cdot \log(|V|)$ advice bits.

1.2 Related Work

Imase and Waxman [8] were the first to study the Steiner Tree problem in the online setting and showed a tight bound of $\Theta(\log(N))$ for its competitive-ratio. Alon and Azar [9] show that almost the same lower bound holds also for the planar case, where the vertices are points in the Euclidean plane. Berman and Coulston [10] and Awerbuch et al. [11] study a generalized version of the problem and related problems. More recently, Garg et al. [12] considered a stochastic version of the online problem.

1.3 Some Notation

For two understood objects a and b, (i.e., instances, paths, etc.) we denote their concatenation by $a \circ b$. All logarithms are to base 2. For a non-empty set S we denote by $x \overset{\mathrm{r}}{\leftarrow} S$ choosing an element x uniformly at random from S. For a positive natural number n, we denote $[n] \overset{\mathrm{def}}{=} \{1, \ldots, n\}$. For a graph $G = (V, E)$ and two vertices $s, t \in V$, we denote by $s \rightsquigarrow t$ a simple path from s to t in G.

2 The Terminal-Greedy Algorithm

In this section, we present an $O(\log(\frac{N}{q}))$-competitive algorithm that utilizes $q' \overset{\mathrm{def}}{=} q \cdot \log(|V|)$ advice bits, for any $q \in [N-1]$.

Observe that an advice of size $(N-1)\log(|V|)$ is always sufficient in order to obtain an optimal solution, since the algorithm is required to make its first decision only upon receiving the second vertex. Therefore, one could canonically encode the rest of the input using $(N-1)\log(|V|)$ bits.

Recall that the online greedy algorithm that connects the current terminal v_i using the shortest weighted path to a vertex from the current solution is $O(\log(N))$-competitive. Our algorithm is obtained by a modification of the greedy algorithm. Whereas the greedy algorithm connects the next vertex to the current solution by using the shortest path to *any* vertex of the current solution, the terminal-greedy algorithm connects a new vertex using a shortest path to one of the terminals of the current input, ignoring possible shorter paths connecting to some non-terminal vertices already chosen by the solution.

The following lemma, whose proof is omitted in this extend abstract, was used in the proof of [9] for the standard greedy algorithm and still holds for our terminal-greedy algorithm.

Lemma 1. *Let* OptVal *denote the value of the optimal solution to an instance. Let* $k \in \mathbb{N}$*. The number of steps in which the terminal-greedy algorithm (without advice) adds edges of total weight more than* $2 \cdot \mathsf{OptVal}/k$ *is at most* $k-1$*.*

The terminal-greedy algorithm utilizes its advice as a list of q vertices from the instance. Intuitively, the vertices given as advice are the q most expensive ones for the input when given in an online fashion (without advice). The challenge is to show that no further costs are incurred by the algorithm using this approach.

Next, we describe the terminal-greedy algorithm with advice of size $q \cdot \log(|V|)$:[1] When the algorithm receives its first terminal vertex v_1 from the instance it computes the optimal (offline) Steiner Tree for the terminal set that consists of the q vertices given as advice along with v_1. Then, it sets the computed tree for this terminal set (which consists of $q+1$ vertices) as the current solution.

For $i \geq 2$, upon receiving a terminal v_i, the algorithm proceeds as follows: If v_i has already appeared in the advice, it does nothing. Otherwise, the algorithm computes the shortest path (in G) from v_i to all the terminals that have previously appeared as part of the instance $(v_1, \ldots, v_{i-1})$ and connects v_i using the shortest path among those $i-1$ paths (and to the lexicographically first in case that there is more than one).

Theorem 1. *Let* $1 \leq q \leq N-1$*. The terminal-greedy algorithm with* $q \cdot \log(|V|)$ *advice bits is* $O(\log(\frac{N}{q}))$*-competitive.*

Proof. Using induction one shows that, in every step, the chosen subgraph is a solution to the current instance. The rest of the proof is concerned with showing the bound on the cost of the algorithm.

[1] Fomally, following the model of Hromkovic et al. [7] the advice string is infinite, and therefore another $2\log(|V|)$ advice bits are given at the beginning of the input, encoding the value q. In our setting this can be ignored, incurring an additive constant imprecision of at most 1.

We show that, for every $q > 0$, there exists a set of size q such that, for every instance with N terminals with optimal (offline) solution of value OptVal, the solution computed by the algorithm has cost at most $O(\mathsf{OptVal} \cdot \log(\frac{N}{q}))$.

For any $v_i \in \{v_1, \ldots, v_N\}$ we denote by $c(v_i)$ the cost incurred when adding vertex v_i according to the terminal-greedy algorithm (without advice). That is, $c(v_i)$ is the sum of the weights of all the edges chosen at step i in order to connect v_i to the solution. Let us sort the vertices of the instance according to their costs. That is, let $[v'_1, v'_2, \ldots, v'_N]$ be the sorted permutation of $[v_1, \ldots, v_N]$, where $c(v'_1) \geq c(v'_2) \geq \cdots \geq c(v'_N)$.

We claim that the terminal-greedy algorithm with advice $[v'_1, \ldots, v'_q]$ is $\log(\frac{N}{q})$-competitive. Indeed, the tree computed by the algorithm after v_1 is received has cost at most OptVal, as the optimal tree is one possible solution to it. Now, whenever a vertex v_i is received, it behaves exactly[2] as the greedy-terminal algorithm without advice would, and therefore its cost for this vertex is $c(v_i)$.

By Lemma 1, we know that for every $i \in [n]$ it holds that $c(v'_i) \leq (2 \cdot \mathsf{OptVal})/i$ as otherwise, since the v_i's are sorted, we get i vertices that each incurs a cost of more than $(2 \cdot \mathsf{OptVal})/i$. Since $c(v_1) = c(v'_n) = 0$, the total cost of the algorithm is bounded by

$$\begin{aligned}
\mathsf{OptVal} + \sum_{i=q+1}^{N-1} c(v'_i) &\leq \mathsf{OptVal} + 2 \cdot \mathsf{OptVal} \sum_{i=q+1}^{N-1} \frac{1}{i} \\
&= \mathsf{OptVal} \left(1 + 2(\sum_{i=1}^{N-1} \frac{1}{i} - \sum_{i=1}^{q} \frac{1}{i}) \right) \\
&< \mathsf{OptVal} \left(1 + \frac{2}{\log(e)} \cdot \log(\frac{N-1}{q}) + \frac{1}{q} \right) \ ,
\end{aligned}$$

where in the last inequality we used the fact that $\sum_{i=1}^{k} \frac{1}{i} = \ln(k) + \gamma + \frac{1}{2k} \pm o(\frac{1}{k})$, where $\gamma \approx 0.5772$ is the Euler-Mascheroni constant. Finally, recall that a subset of the vertices of size q can be described using $q \cdot \log(|V|)$ bits. The bound follows. □

3 A Matching Lower Bound

In this section we show a lower bound matching the competitive ratio guarantee of the algorithm presented in Section 2. As mentioned, our construction holds already for the unweighted case where $w(e) = 1$, thus we omit w from our notation.

3.1 Edge-Efficient Algorithms

It will be useful for us to analyze the performance of algorithms that enjoy a canonical structure and have some guarantees on their behavior. We identify such

[2] Note that in general this does not hold for the 'standard' greedy-algorithm.

a class of algorithms next. An online algorithm A for ST is edge-efficient if, for every instance $\mathcal{I}$, when removing any edge from the solution $A(\mathcal{I})$, the resulting graph is not a solution. That is, removing any edge from $A(\mathcal{I})$ disconnects two terminals $v, v' \in \mathcal{I}$.

The next lemma shows that edge-efficient algorithms are as powerful as general algorithms and therefore we can focus our analysis on them. The proof of the following lemma is omitted in this extend abstract.

Lemma 2. *For every deterministic online algorithm A for ST there exists an edge-efficient algorithm A' such that, for every instance $\mathcal{I}$, we have $cost(A'(\mathcal{I})) \leq cost(A(\mathcal{I}))$.*

3.2 Diamond Graphs and Our Instance Distribution

For vertices s and t and a list of natural numbers $[\ell_1, \ell_2, \ldots, \ell_n]$, we define the diamond graph of level n on vertices s and t, denoted $D_n[\ell_1, \ldots, \ell_n](s,t)$, recursively as follows:

1. The graph $D_0[\,](s,t)$ (of level $n = 0$ with an empty list) consists of the vertices s and t and the single edge (s,t).
2. Given $G(s',t') \stackrel{\text{def}}{=} D_n[\ell_1, \ldots, \ell_n](s',t')$, a diamond graph of level n on vertices s', t', the graph $D_{n+1}[z, \ell_1, \ell_2, \ldots, \ell_n](s,t)$ is constructed as follows: We start with the vertices: s, t and $m_1, \ldots, m_z$. Next, we construct the following $2z$ copies of $G(s',t')$: $G(s,m_1), \ldots, G(s,m_z)$ and $G(m_1,t), \ldots, G(m_z,t)$, where $G(x,y)$ is a copy of the graph $G(s',t')$, where the vertices s' and t' are identified with x and y. Finally, the resulting graph is the union of the $2z$ diamond graphs $G(s,m_1), \ldots, G(s,m_z), G(m_1,t), \ldots, G(m_z,t)$.

We call the parameter ℓ_i the width of level i of the graph. and the vertices $m_1, \ldots, m_{\ell_1}$ the middle vertices of $D_n[\ell_1, \ldots, \ell_n](s,t)$. Note that the graphs in the union are almost disjoint, that is, any two of them share at most one vertex (and no edges).

For a fixed $n \in \mathbb{N}$ our instance distribution generates simultaneously an instance $\mathcal{I}$ that contains $N + 1 = 2^n + 1$ terminals, and a path P from s to t of length $N = 2^n$, which is an optimal solution to it.[3] The first two vertices are always s and t, and vertices along the path are chosen level by level, where choosing the vertices of level $i+1$ can be thought of as a refinement of the path along the vertices of level i. The idea is that the algorithm has to connect all the level-i vertices before level-$(i+1)$ vertices are revealed. Formally, the instance of $\mathsf{ST}(D_n[\ell_1, \ldots, \ell_n](s,t))$ is generated according to Process 1.

The following propositions follow by simple induction and the definitions of $D_n[\ell_1, \ldots, \ell_n](s,t)$, GenerateInstance and GeneratePath.

Proposition 1. *The graph $D_n[\ell_1, \ldots, \ell_n](s,t)$ contains $2^n \cdot \prod_{i=1}^{n} \ell_i$ edges.*

[3] For simplicity of presentation, we use an instance of $N+1$ instead of N terminals.

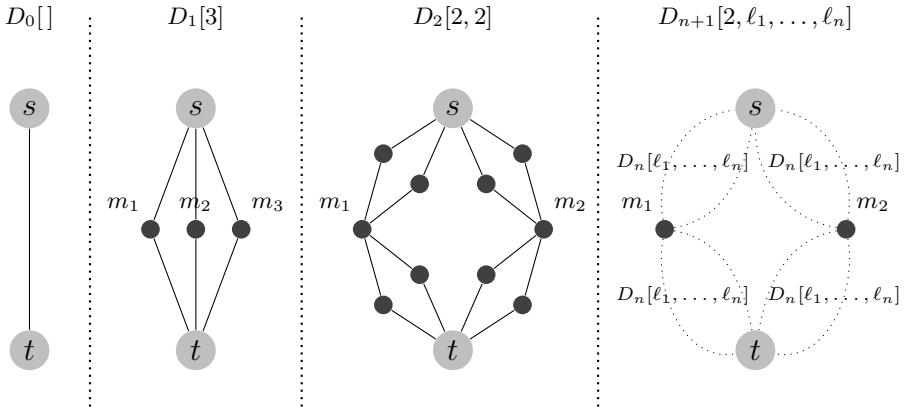

Fig. 1. Diamond Graphs

Process 1. GENERATEINSTANCE

Input: A graph $D_n[\ell_1,\ldots,\ell_n](s,t)$
Output: An instance $\mathcal{I}$ of $\mathsf{ST}(D_n[\ell_1,\ldots,\ell_n](s,t))$

$\mathcal{I} \leftarrow [s]$ ▷ Every instance starts with the vertex s
$\mathcal{I} \leftarrow \mathcal{I} \circ [t]$ ▷ followed by the vertex t.
$P \leftarrow$ GENERATEPATH$(D_n[\ell_1,\ldots,\ell_n](s,t))$

procedure GENERATEPATH$(D_k[\ell'_1,\ldots,\ell'_k](u,v))$
  if $k = 0$ **then**
    return $e = (u,v)$
  else
    Choose $x \xleftarrow{r} \{m_1,\ldots,m_{\ell'_1}\}$ ▷ $m_1,\ldots,m_{\ell'_1}$ are the middle vertices of
    $\mathcal{I} \leftarrow \mathcal{I} \circ [x]$ ▷ $D_k[\ell'_1,\ldots,\ell'_k](u,v)$
    $P_1 \leftarrow$ GENERATEPATH$(D_{k-1}[\ell'_2,\ldots,\ell'_k](u,x))$
    $P_2 \leftarrow$ GENERATEPATH$(D_{k-1}[\ell'_2,\ldots,\ell'_k](x,v))$
    return $P_1 \circ P_2$
  end if
end procedure

Proposition 2. *Let $n \geq 1$. A simple path $s \rightsquigarrow t$ on $D_n[\ell_1, \ldots, \ell_n](s,t)$ is of the form $s \rightsquigarrow x \rightsquigarrow t$ for some $x \in \{m_1, \ldots, m_{\ell_1}\}$ and contains exactly 2^n edges.*

Proposition 3. *The path P computed during the execution of* GenerateInstance$(D_n[\ell_1, \ldots, \ell_n](s,t))$ *is a solution to the generated instance that contains exactly 2^n edges.*

Proposition 4. *During the run of* GeneratePath$(D_k[\ell'_1, \ldots, \ell'_k](u,v))$*, when the algorithm adds a vertex x as the next vertex of the instance $\mathcal{I}$ (Line 10), both u and v have already appeared in $\mathcal{I}$ and no other vertex from $D_k[\ell'_1, \ldots, \ell'_k](u,v)$ is contained in $\mathcal{I}$.*

Lemma 3. *Consider an execution of* GenerateInstance$(D_n[\ell_1, \ldots, \ell_n](s,t))$*, and let A be an edge-efficient algorithm. The number of edges added to the solution by A during every call to* GeneratePath$(D_k[\ell'_1, \ldots, \ell'_k](u,v))$ *is at least $W_k \stackrel{\text{def}}{=} \sum_{i=1}^{k} \left(\frac{2^k}{2^i} \sum_{j=1}^{2^{i-1}} X_{i,j} \right)$, where the $X_{i,j}$'s are independent Bernoulli random variables with $\Pr[X_{i,j} = 0] = 1/\ell'_i = 1 - \Pr[X_{i,j} = 1]$.*

Before proving the lemma, we prove the following proposition on the structure of the current solution restricted to the subgraph $D_k[\ell'_1, \ldots, \ell'_k](u,v)$ when GeneratePath(u,v) is called.

Proposition 5. *Let $k \in \{1, \ldots, n\}$ and let A be an edge-efficient algorithm. Consider an execution of* GeneratePath$(D_n[\ell_1, \ldots, \ell_n](s,t))$*. Whenever a call* GeneratePath$(D_k[\ell'_1, \ldots, \ell'_k](u,v))$ *is made, either (1) the current solution chosen by A restricted to the subgraph $D_k[\ell'_1, \ldots, \ell'_k](u,v)$ contains no edges, or, (2) $D_k[\ell'_1, \ldots, \ell'_k](u,v)$ contains a simple path of the form $u \rightsquigarrow y \rightsquigarrow v$ for some $y \in \{m_1, \ldots, m_{\ell'_1}\}$, where $m_1, \ldots, m_{\ell'_1}$ are the middle vertices $D_k[\ell'_1, \ldots, \ell'_k](u,v)$, and no other edges.*

Proof. By Proposition 4, we know that the vertices u and v have already appeared in the instance, and therefore they are connected in the current solution, and, in particular, by some simple path $u \rightsquigarrow v$. Consider the first edge (u,z) of this path. If z is contained in $D_k[\ell'_1, \ldots, \ell'_k](u,v)$, then, since the only way to reach a vertex outside of $D_k[\ell'_1, \ldots, \ell'_k](u,v)$ is through the vertices u and v, the entire path is contained in $D_k[\ell'_1, \ldots, \ell'_k](u,v)$. Conversely, if z is not contained in $D_k[\ell'_1, \ldots, \ell'_k](u,v)$, by the same argument, it follows that the path $u \rightsquigarrow v$ does not contain any inner vertex of $D_k[\ell'_1, \ldots, \ell'_k](u,v)$.

We argue that in both cases no other edges of the current solution are incident to $D_k[\ell'_1, \ldots, \ell'_k](u,v)$. Assume the contrary, and let $e \in D_k[\ell'_1, \ldots, \ell'_k](u,v)$ be such an edge (not in $u \rightsquigarrow v$ in the first case and any edge in $D_k[\ell'_1, \ldots, \ell'_k](u,v)$ in the second case).

Observe that, since e is an internal edge of $D_k[\ell'_1, \ldots, \ell'_k](u,v)$ not on the path $u \rightsquigarrow v$ and since by Proposition 4 at this point no other internal vertex is chosen to the current instance, the vertices of the current instance remain connected after removing e. This contradicts the edge-efficiency property of the current solution chosen by A. □

Proof (of Lemma 3). We use induction on k, the parameter of the diamond subgraph. For $k = 0$, the claim holds trivially since $W_0 = 0$ and at least zero edges are added. Let $k > 0$, and assume that the claim holds for all $k' < k$. Let GeneratePath$(D_k[\ell'_1, \ldots, \ell'_k](u, v))$ be a call made during the execution of GenerateInstance$(D_n[\ell_1, \ldots, \ell_n](s, t))$. By Proposition 5, the solution chosen limited to $D_k[\ell'_1, \ldots, \ell'_k](u, v)$ either (1) has no edges, or, (2) has exactly one simple path between u and v, which by Proposition 2 has the form $u \rightsquigarrow y \rightsquigarrow v$, for $y \in \{m_1, \ldots, m_{\ell'_1}\}$. Without loss of generality, we assume that the path is of the form $u \rightsquigarrow m_1 \rightsquigarrow v$. In the first case, after lines 9 and 10, the algorithm connects the vertex x to the graph, which must be via the vertex u or v, in which case $\frac{2^k}{2}$ edges are added. In the second case, with probability $1 - 1/\ell'_1$ the vertex x is chosen from $m_2, \ldots, m_{\ell'}$, in which case as before, it must be connected using a path from u or v, which adds $\frac{2^k}{2}$ edges.

We conclude that the number of edges added to the solution due to the choice of the vertex x is at least $\frac{2^k}{2} X_{1,1}$, where $X_{1,1}$ is distributed according to $\Pr[X_{1,1} = 0] = 1/\ell'_1 = 1 - \Pr[X_{1,1} = 1]$.

Additionally, using the inductive hypothesis, the algorithm adds $W'_{k-1} = \sum_{i=1}^{k-1} \left(\frac{2^{k-1}}{2^i} \sum_{j=1}^{2^{i-1}} X'_{i,j} \right)$ and $W''_{k-1} = \sum_{i=1}^{k-1} \left(\frac{2^{k-1}}{2^i} \sum_{j=1}^{2^{i-1}} X''_{i,j} \right)$ edges during the executions of GeneratePath$(D_{k-1}[\ell'_2, \ldots, \ell'_k](s, x))$ and GeneratePath$(D_{k-1}[\ell'_2, \ldots, \ell'_k](x, t))$, respectively, where $X'_{i,j}$ and $X''_{i,j}$ are Bernoulli random variables distributed according to $\Pr[X'_{i,j} = 0] = \Pr[X''_{i,j} = 0] = 1/\ell'_{i+1} = 1 - \Pr[X''_{i,j} = 1] = 1 - \Pr[X'_{i,j} = 1]$.

Moreover, since the random choices of GeneratePath are independent, we have that the Bernoulli random variables are independent. Setting $X_{i+1,j} \stackrel{\text{def}}{=} X'_{i,j}$ and $X_{i+1,2^{i-1}+j} \stackrel{\text{def}}{=} X''_{i,j}$ for all $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, 2^{i-1}\}$, we obtain that during the execution of GeneratePath the algorithm adds at least $\frac{2^k}{2} X_{1,1} + W'_{k-1} + W''_{k-1} = \sum_{i=1}^{k} \left(\frac{2^k}{2^i} \sum_{j=1}^{2^{i-1}} X_{i,j} \right)$ edges to the solution. The lemma is proved. □

Corollary 1. *Any deterministic algorithm A for* ST*, when given an instance generated by* GeneratePath$(D_n[\ell_1, \ldots, \ell_n](s, t))$*, outputs a solution that contains at least*

$$\sum_{i=1}^{\log(N)} \left(\frac{N}{2^i} \sum_{j=1}^{2^{i-1}} X_{i,j} \right)$$

edges, where the $X_{i,j}$'s are as in Lemma 3.

Proof. By Proposition 2, we may assume that A is an edge-efficient algorithm. The corollary follows by Lemma 3. □

We refer to an edge added due to some $X_{i,j} = 1$ as an edge of level i and say that in this case the algorithm made a wrong choice on $X_{i,j}$. Indeed, in this case it was possible to connect some vertices u and v through a middle vertex m such that the algorithm would not have had to add edges due to $X_{i,j}$.

3.3 Deriving the Lower Bound

In this section, we show that for every algorithm with advice size $q \cdot \log(|V|)$ the terminal-greedy algorithm is best possible.

The input distribution we use is a diamond graph with parameters that depend on the advice length of the specific algorithm it seeks to fail. Consider an algorithm taking $N \cdot 2^{-j(N)} \cdot \log(|V|)$ advice bits, where $\log(N) \geq j(N) \geq 0$.

We can assume that $\log(N) \geq j(N) > 10$ and, furthermore, that $j(N)$ is an even integer number. The first assumption is trivial to satisfy, since every algorithm is at most strictly 1-competitive and so for a constant $j(N)$ the asymptotic bound already holds. The second assumption incurs an additive term of 2 (recall that the bound we show is logarithmic). Therefore, both assumptions are made without loss of generality.

Set $j'(N) \stackrel{\text{def}}{=} \frac{j(N)}{2}$ and consider $D_n[\ell_1, \ldots, \ell_n](s,t)$, the diamond graph with $\log(N)$ levels, where the first $\log(N) - j'(N)$ levels are of width 2 and the last $j'(N)$ levels are of width N^2. That is, $\ell_1 = \cdots = \ell_{\log(N)-j'(N)} = 2$ and $\ell_{\log(N)-j'(N)+1} = \cdots = \ell_{\log(N)} = N^2$. For the rest of this section we refer to this graph as G.

We can show that for every online algorithm with $q \cdot \log(|V|)$ advice bits there exists an input on which it does not perform better than $\Omega(\log(\frac{N}{q}))$ compared to an optimal offline solution:

Theorem 2. *Let A be an online algorithm for* ST *taking $q' \stackrel{\text{def}}{=} q \cdot \log(|V|)$ advice bits, where $q \stackrel{\text{def}}{=} N \cdot 2^{-j(N)}$ and $\log(N) \geq j(N) \geq 0$. Then A has a competitive ratio of at least $\Omega(\log(\frac{N}{q}))$.*

We present an overview of the proof. Recall that for a fixed advice string $\phi \in \{0,1\}^{q'}$, the algorithm A "hard-wired" with ϕ (denoted A_ϕ) is a deterministic online algorithm and therefore Corollary 1 establishes that on a random instance of GenerateInstance($D_n[\ell_1, \ldots, \ell_n](s,t)$) it chooses at least W_n edges.

We show that, a solution for an instance chosen by GenerateInstance contains, with very high probability, roughly $N \cdot \log(\frac{N}{q})$ edges, and then use the union bound to show that there exists an instance that makes all of the $2^{q'}$ algorithms choose this number of edges.

Using the machinery developed for general diamond graphs and the properties of GenerateInstance we show that, by our choice of $j'(N)$, it holds that $\log(|V|)$ is not too large, and for each of the last $j'(N)$ levels of the graph, a fixed deterministic algorithm chooses a linear (in N) number of edges with probability roughly $2^{-q'} = 2^{-q \cdot \log(|V|)}$.

Finally, we use the probabilistic method and show that there exists an instance on G, such that every A_ϕ chooses many edges on every level τ of the last $j'(N)$ levels in G.

Proof. Using Proposition 1 we have that the number of edges in G is $2^n \cdot 2^{(n-j'(N))} \cdot (N^2)^{j'(N)} < 4^n \cdot (N^2)^{j'(N)}$. Since the number of vertices in a graph is at most twice the number of its edges, we obtain that $|V|$, the number of vertices

in G, is at most $2 \cdot 4^n \cdot (N^2)^{j'(N)}$, and therefore $\log(|V|) < 4 \cdot \log(N) \cdot j'(N)$. Therefore, we can bound the advice size by

$$q' = 2^{n-j(N)} \cdot \log(|V|) < 2^{n-j(N)} \cdot 4 \cdot \log(N) \cdot j'(N) = 4 \cdot 2^{n-2 \cdot j'(N)} \cdot n \cdot j'(N) \ . \quad (1)$$

By Lemma 3 and Corollary 1, for every level τ, where $n - j'(N) < \tau \leq n$, the probability that an edge-efficient deterministic algorithm is correct on at least $\frac{2^{\tau-1}}{4}$ of its choices for level τ (i.e., at least this number of $X'_{\tau,j}s$ are 0) can be computed as

$$\begin{aligned}
\Pr\left[\exists S \subset \left[2^{\tau-1}\right] : |S| = \frac{2^{\tau-1}}{4} \wedge \forall p \in S : X_{\tau,p} = 0\right] &< \binom{2^{\tau-1}}{\frac{2^{\tau-1}}{4}} \cdot \left(\frac{1}{N^2}\right)^{\frac{2^{\tau-1}}{4}} \\
&\leq \left(2^{\tau-1}\right)^{\frac{2^{\tau-1}}{4}} \cdot \left(\frac{1}{N^2}\right)^{\frac{2^{\tau-1}}{4}} \\
&= \left(\frac{2^{\tau-1}}{2^{2n}}\right)^{\frac{2^{\tau-1}}{4}} \\
&\leq \left(\frac{1}{2^n}\right)^{\frac{2^{\tau-1}}{4}} \\
&= 2^{-\left(\frac{n \cdot 2^{\tau-1}}{4}\right)} \\
&\leq 2^{-\left(\frac{n \cdot 2^{n-j'(N)}}{4}\right)} \ .
\end{aligned}$$

Next we apply the union bound twice: The probability p that there exists a level $n - j'(N) < \tau \leq n$ for which one of the $2^{q'}$ deterministic algorithms makes more than $\frac{2^{\tau-1}}{4}$ correct choices can be bounded as follows:

$$\begin{aligned}
p &< 2^{q'} \cdot j'(N) \cdot 2^{-\left(\frac{n \cdot 2^{n-j'(N)}}{4}\right)} && (2)\\
&< 2^{\left(4 \cdot 2^{(n-2 \cdot j'(N))} \cdot n \cdot j'(N)\right)} \cdot 2^{\log(j'(N))} \cdot 2^{-\left(\frac{n \cdot 2^{n-j'(N)}}{4}\right)} && (3)\\
&< 2^{\left(5 \cdot 2^{(n-2 \cdot j'(N))} \cdot n \cdot j'(N)\right)} \cdot 2^{-\left(\frac{n \cdot 2^{n-j'(N)}}{4}\right)} && (4)\\
&= 2^{\left(\left(n \cdot 2^{n-j'(N)}\right)\left(5 \cdot 2^{-j'(N)} \cdot j'(N) - \frac{1}{4}\right)\right)} < 1 && (5)
\end{aligned}$$

In turn, observe that this implies that there exists a fixed instance $\mathcal{I}'$ such that the algorithm A, for every choice of advice of length q', and for every level τ in the range, makes at least $\frac{3 \cdot 2^{\tau-1}}{4}$ incorrect choices, each of which results in an addition of $\frac{N}{2^\tau}$ edges by the algorithm. Therefore, for this instance, the algorithm chooses a solution that contains at least

$$\sum_{\tau=\log(N)-j'(N)+1}^{\log(N)} \frac{N}{2^\tau} \cdot \frac{3 \cdot 2^{\tau-1}}{4} = \frac{3}{8} \cdot N \cdot j'(N) \in \Omega(N \cdot j(N)) = \Omega\left(N \cdot \log(\frac{N}{q})\right)$$

edges.

On the other hand, recall that, by Proposition 3, since $\mathcal{I}'$ is just one of the possible instances generated by GenerateInstance, there exists a solution that consists of N edges. The lower bound of $\Omega(\log(\frac{N}{q}))$ on the competitive ratio follows. □

Acknowledgments. We would like to thank Juraj Hromkovič for suggesting this research direction and Hans-Joachim Böckenhauer for his very helpful comments and suggestions regarding the presentation of this work. We thank the anonymous reviewers for their helpful comments.

References

1. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM 28(2), 202–208 (1985)
2. Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. Cambridge University Press (1998)
3. Böckenhauer, H.-J., Komm, D., Královič, R., Rossmanith, P.: On the advice complexity of the knapsack problem. In: Fernández-Baca, D. (ed.) LATIN 2012. LNCS, vol. 7256, pp. 61–72. Springer, Heidelberg (2012)
4. Dobrev, S., Královič, R., Pardubská, D.: How much information about the future is needed? In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 247–258. Springer, Heidelberg (2008)
5. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R., Mömke, T.: On the advice complexity of online problems. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 331–340. Springer, Heidelberg (2009)
6. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. Theor. Comput. Sci. 412(24), 2642–2656 (2011)
7. Hromkovič, J., Královič, R., Královič, R.: Information complexity of online problems. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 24–36. Springer, Heidelberg (2010)
8. Imase, M., Waxman, B.M.: Dynamic steiner tree problem. SIAM J. Discrete Math. 4(3), 369–384 (1991)
9. Alon, N., Azar, Y.: On-line steiner trees in the euclidean plane. In: Symposium on Computational Geometry, pp. 337–343 (1992)
10. Berman, P., Coulston, C.: On-line algorithms for steiner tree problems (extended abstract). In: Leighton, F.T., Shor, P.W. (eds.) STOC, pp. 344–353. ACM (1997)
11. Awerbuch, B., Azar, Y., Bartal, Y.: On-line generalized steiner problem. Theor. Comput. Sci. 324(2-3), 313–324 (2004)
12. Garg, N., Gupta, A., Leonardi, S., Sankowski, P.: Stochastic analyses for online combinatorial optimization problems. In: Teng, S.H. (ed.) SODA, pp. 942–951. SIAM (2008)