Call Admission Problems on Trees with Advice
(Extended Abstract)

Hans-Joachim Böckenhauer(✉), Nina Corvelo Benz, and Dennis Komm

Department of Computer Science, ETH Zurich, Zurich, Switzerland
{hjb,dennis.komm}@inf.ethz.ch, cnina@student.ethz.ch

Abstract. A well-studied problem in the online setting, where requests have to be answered immediately without knowledge of future requests, is the call admission problem. In this problem, we are given nodes in a communication network that request connections to other nodes in the network. A central authority may accept or reject such a request right away, and once a connection is established its duration is unbounded and its edges are blocked for other connections. This paper examines the admission problem in tree networks. The focus is on the quality of solutions achievable in an advice setting, that is, when the central authority has some information about the incoming requests. We show that $O(m \log d)$ bits of additional information are sufficient for an online algorithm run by the central authority to perform as well as an optimal offline algorithm, where m is the number of edges and d is the largest degree in the tree. In the case of a star tree network, we show that $\Omega(m \log d)$ bits are also necessary (note that $d = m$). Additionally, we present a lower bound on the advice complexity for small constant competitive ratios and an algorithm whose competitive ratio gradually improves with added advice bits to $2\lceil \log_2 n \rceil$, where n is the number of nodes.

1 Introduction

A well-studied problem in the context of regulating the traffic in communication networks is the so-called *call admission problem*, where a central authority decides about which subset of communication requests can be routed. This is a typical example of an *online problem*: every request has to be routed or rejected immediately without the knowledge about whether some forthcoming, possibly more profitable, requests will be blocked by this decision.

We consider the *call admission problem on trees* (short CAT), which is an online maximization problem. An instance $I = (r_1, \ldots, r_k)$ consists of requests $r = (v_i, v_j)$ with $i, j \in \{0, \ldots, n-1\}$ and $v_i < v_j$, representing the unique path in a tree network that connects vertices v_i and v_j. We require that all requests in I are pairwise distinct. The first request contains the network tree $T = (V, E)$, given to the algorithm in form of an adjacency list or matrix. In particular, we study the problem framework in which an accepted connection has an unbounded

C. J. Colbourn et al. (Eds.): IWOCA 2019, LNCS 11638, pp. 108–121, 2019.
https://doi.org/10.1007/978-3-030-25005-8_10

duration and each edge in the network may be used by at most one request, i.e., it has a capacity of 1. Thus, a valid solution $O = (y_1, \ldots, y_k) \in \{0,1\}^k$ for I describes a set $\mathcal{P}(I,O) \coloneqq \{r_i \mid i \in \{1,\ldots,k\} \text{ and } y_i = 1\}$ of edge-disjoint paths in T, where $\text{gain}(I,O) \coloneqq |\mathcal{P}(I,O)|$. Whenever I is clear from the context, we write $\text{gain}(O)$ instead of $\text{gain}(I,O)$.

An *online algorithm* ALG for CAT computes the output sequence (solution) $\text{ALG}(I) = (y_1, \ldots, y_k)$, where y_i is computed from $x_1, \ldots, x_i$. The gain of ALG's solution is given by $\text{gain}(\text{ALG}(I))$. ALG is *c-competitive*, for some $c \geq 1$, if there exists a constant γ such that, for every input sequence I, $\text{gain}(\text{OPT}(I)) \leq c \cdot \text{gain}(\text{ALG}(I)) + \gamma$, where OPT is an optimal offline algorithm for the problem. This constitutes a measure of performance used to compare online algorithms based on the quality of their solutions, which was introduced by Sleator and Tarjan [18].

The downside of *competitive analysis* as a measurement of performance is that it seems rather unrealistic to compare the performance of an all-seeing offline algorithm to that of an online algorithm with no knowledge at all about future requests. This results in this method not really apprehending the hardness of online computation. Moreover, it cannot model information about the input that we may have outside the strictly defined setting of the problem. The advice model was introduced as an approach to investigate the amount of information about the future an online algorithm lacks [6,7,12,13,15]. It investigates how many bits of information are necessary and sufficient to achieve a certain output quality, which has interesting implications for, e.g., randomized online computation [5,9,16]. For lower bounds on this number in particular, we do not make any assumptions on the kind of information the advice consists of.

Let Π be an online maximization problem, and consider an input sequence $I = (x_1, \ldots, x_k)$ of Π. An *online algorithm* ALG *with advice* computes the output sequence $\text{ALG}(I)^\phi = (y_1, \ldots, y_k)$ such that y_i is computed from $\phi, x_1, \ldots, x_i$, where ϕ is the content of the advice tape, i.e., an infinite binary sequence. ALG is *c-competitive with advice complexity* $b(k)$ if there exists a constant γ such that, for every k and for each input sequence I of length at most k, there exists some ϕ such that $\text{gain}(\text{OPT}(I)) \leq c \cdot \text{gain}(\text{ALG}^\phi(I)) + \gamma$ and at most the first $b(k)$ bits of ϕ have been read during the computation of $\text{ALG}^\phi(I)$.

For a better understanding, consider the following example. A straightforward approach to an optimal online algorithm with advice for CAT is to have one bit of advice for each request in the given instance. This bit indicates whether the request should be accepted or not. Thus, ALG reads $|I|$ advice bits and accepts only requests in $\text{OPT}(I)$, i.e., ALG is optimal.

This approach gives us a bound on the advice complexity that is linear in the size of the instance. Opposed to most other online problems however, call admission problems, like CAT, are usually analyzed with respect to the size of the communication network instead of the size of an instance as stated in the general definition. Thus, the advice complexity of this naive optimal algorithm on a tree with n vertices is of order n^2.

Related Work. The call admission problem is a well-studied online problem; for an overview of results regarding classical competitive analysis for this problem on various graph topologies, see Chapter 13 in the textbook by Borodin and El-Yaniv [3]. For the call admission problem on path networks (also called the *disjoint path allocation problem*, short DPA), Barhum et al. [2] showed that $l-1$ advice bits are both sufficient and necessary for an online algorithm to be optimal, where l is the length of the path. They also generalized the $\log_2 l$-competitive randomized algorithm for DPA presented by Awerbuch et al. [1]. Gebauer et al. [14] proved that, with $l^{1-\varepsilon}$ bits of advice, no online algorithm for DPA is better than $(\delta \log_2 l)/2$-competitive, where $0 < \delta < \varepsilon < 1$. The advice complexity of call admission problems on grids was investigated by Böckenhauer et al. [8].

When considering trees as network structure, we still have the property that the path between two nodes is unique in the network, thus all lower bounds on the advice complexity easily carry over by substituting the length l of the path network by the diameter D of the tree network. These lower bounds can be further improved as shown in Sect. 2. Concerning upper bounds, Borodin and El-Yaniv [3] presented two randomized online algorithms, a $2\lceil \log_2 n \rceil$-competitive algorithm, first introduced by Awerbuch et al. [1], and an $O(\log D)$-competitive algorithm. We will modify the former in Subsect. 3.2 to an online algorithm that reads $\lceil \log_2 \log_2 n - \log_2 p \rceil$ advice bits and is $((2^{p+1}-2)\lceil \log_2 n/p \rceil)$-competitive, for any integer $1 \leq p \leq \log_2 n$.

Another problem closely related to DPA is the length-weighted disjoint path allocation problem on path networks, where instead of optimizing the number of accepted requests, one is interested in maximizing the combined length of all accepted requests. Burjons et al. [10] extensively study the advice complexity behavior of this problem.

Overview. In Sect. 2, we present the already mentioned lower bound for optimality, which even holds for star trees. We complement this with lower bounds for the trade-off between the competitive ratio and advice, based on reductions from the well-known string guessing problem [4]. Section 3 is devoted to the corresponding upper bounds. In Subsect. 3.1, we present algorithms for computing an optimal solution, both for general trees and for star trees and k-ary trees. As mentioned above, in Subsect. 3.2, we analyze the trade-off between the competitive ratio and advice and estimate how much the competitive ratio degrades by using less and less advice bits. Due to space restrictions, some of the proofs are omitted in this extended abstract.

Notation. Following common conventions, m is the number of edges in a graph and n the number of vertices. The degree of a vertex v is denoted by $d(v)$. Let $v_0, \ldots, v_{n-1}$ be the vertices of a tree T with some order $v_0 < \cdots < v_{n-1}$. This order can be arbitrarily chosen, but is fixed and used as order in the adjacency matrix or adjacency list of the tree. Hence, an algorithm knows the ordering on the vertices when given the network.

For the sake of simplicity, we sometimes do not enforce that $v < v'$, but regard (v, v') and (v', v) as the same request. For a request $r = (v, v')$, the function edges: $V \times V \to \mathcal{P}(E)$ returns, for request r, the set of edges corresponding to

the unique path in T that connects v and v'. Let $\text{edges}(r) \coloneqq \{e_1, \ldots, e_l\}$; we call l the length of request r. Note that all logarithms in this paper are of base 2, unless stated otherwise.

2 Lower Bounds

First we present lower bounds on the number of advice bits for the call admission problem on trees. We first look at optimal algorithms, then we focus on the connection between the competitive ratio and the advice complexity.

2.1 A Lower Bound for Optimality

Barhum et al. [2] proved that solving DPA optimally requires at least $l-1$ advice bits. As DPA is a subproblem of the call admission problem on trees, this bound also holds for CAT. We can improve on this by considering instances on trees of higher degree. We focus on the simplest tree of high degree, the star tree.

Theorem 1. *There is no optimal online algorithm with advice for CAT that uses less than* $\lceil (m/2)\log(m/\mathrm{e})\rceil$ *advice bits on trees of* m *edges.*

Proof Sketch. This proof is based on the partition-tree method as introduced by Barhum et al. [2]. A partition tree of a set of instances $\mathcal{I}$ is defined as a labeled rooted tree such that (i) every of its vertices v is labeled by a set of input sequences $\mathcal{I}_v$ and a number ϱ_v such that all input sequences in $\mathcal{I}_v$ have a common prefix of length at least ϱ_v, (ii) for every inner vertex v of the tree, the sets at its children form a partition of $\mathcal{I}_v$, and (iii) the root r satisfies $\mathcal{I}_r = \mathcal{I}$. If we consider two vertices v_1 and v_2 in a partition tree that are neither an ancestor of each other, with their lowest common ancestor v and any input instances $I_1 \in \mathcal{I}_{v_1}$ and $I_2 \in \mathcal{I}_{v_2}$ such that, for all optimal solutions for I_1 and I_2, their prefixes of length ϱ_v differ, then any optimal online algorithm with advice needs a different advice string for each of the two input sequences I_1 and I_2. This particularly implies that any optimal online algorithm with advice requires at least $\log(w)$ advice bits, where w is the number of leaves of the partition tree. We sketch the construction of the instances that can be used for building such a partition tree.

Consider the star trees S_{2k} for $k \geq 1$ with $2k$ edges. Let $v_0, v_1, \ldots, v_{2k}$ be the vertices in S_{2k}, where v_0 denotes the center vertex. We construct a set $\mathcal{I}$ of input sequences for S_{2k} so that any two input sequences share a common prefix of requests and each input $I \in \mathcal{I}$ has a unique optimal solution $\textsc{Opt}(I)$, which is only optimal for this particular instance.

Each input instance will be partitioned into k phases. At the end of each of these phases, one vertex will be blocked for all subsequent phases. We can uniquely describe each of our instances $I_{(j_1,\ldots,j_k)}$ by the sequence of these blocked vertices $(v_{j_1}, \ldots, v_{j_k})$. Note that we will not use every possible vertex sequence for our construction. In phase i with $i \in \{1, \ldots, k\}$, we request in ascending order all paths from the non-blocked vertex v_i^* with the smallest index to all other non-blocked vertices, then we block v_i^* and v_{j_i} for all future phases. We note that,

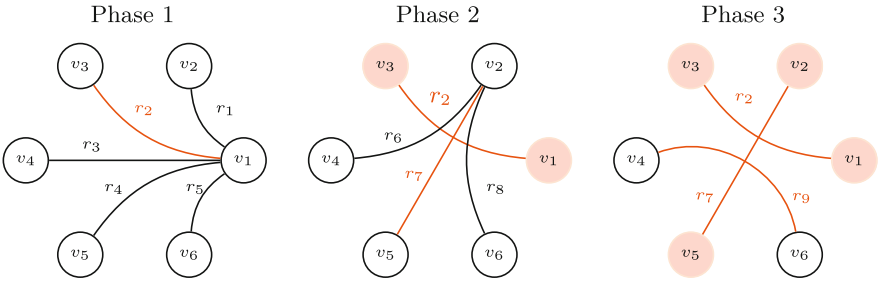

Fig. 1. Input sequence $I_{(3,5,6)} = (r_1, \ldots, r_9)$ on star graph S_6 partitioned into 3 phases. Red vertices are blocked, a red edge indicates the request is in the optimal solution $\textsc{Opt}(I_{(3,5,6)})$. For the sake of simplicity, the center vertex v_0 is omitted from the drawings; thus all lines represent paths of length 2 (Color figure online).

at the start of phase 1, all vertices are non-blocked. Let $\mathcal{I}_{(j_1,\ldots,j_i)}$ denote the set of all input sequences whose tuples have prefix $(j_1, \ldots, j_i)$ with $i \leq k$. Observe that, by definition, all input sequences in $\mathcal{I}_{(j_1,\ldots,j_i)}$ have the same requests until phase i ends and that tuple $(j_1, \ldots, j_k)$ describes exactly one input sequence in $\mathcal{I}$, i.e., $|\mathcal{I}_{(j_1,\ldots,j_k)}| = 1$. Figure 1 shows an illustration of such an input sequence for the star S_6, i.e., for $k = 3$.

We can prove that, for each input sequence $I_{(j_1,\ldots,j_k)}$, the unique optimal solution is $\textsc{Opt}(I_{(j_1,\ldots,j_k)}) := \{r_1, \ldots, r_k\}$, where $r_i := (v_i^*, v_{j_i})$ is the request accepted by Opt in each phase i. The next step is to show that, for any two input sequences in $\mathcal{I}$, their unique optimal solution Opt differs. Consider two input sequences $I, I' \in \mathcal{I}$ with $I \neq I'$ and let $(j_1, \ldots, j_k)$ and $(j'_1, \ldots, j'_k)$ be their identifying tuples, respectively. As the two input sequences are non-identical, there must exist some smallest index i so that $j_i \neq j'_i$. In particular, since i marks the first phase at whose end different vertices are blocked in I and I', the requests in phase i must be identical in both input sequences. Let v_i^* be the non-blocked vertex with the smallest index in phase i in both input sequences. By definition of Opt, request $(v_i^*, v_{j_i}) \in \textsc{Opt}(I)$ and request $(v_i^*, v_{j'_i}) \in \textsc{Opt}(I')$ with $j_i \neq j'_i$. It thus follows that $\textsc{Opt}(I) \neq \textsc{Opt}(I')$ as both requests share an edge.

Since the optimal solution for the common prefix differs between two instances, no online algorithm without advice can be optimal on this set, because, with no additional information on the given instance (i.e., based on the prefix alone) the two instances cannot be distinguished. It follows that the algorithm needs a unique advice string for each instance in the set. Thus, it only remains to bound the number of instances in $\mathcal{I}$. Each instance has a unique label $\mathcal{I}_{(j_1,\ldots,j_k)}$, so that the total number equals the number of tuples $(j_1, \ldots, j_k)$ of pairwise distinct vertex indices, that is,

$$(2k-1) \cdot (2k-3) \cdot (2k-5) \cdot \cdots \cdot 1 = (2k-1)!! = \frac{(2k)!}{2^k \cdot k!},$$

which we can bound from below using Stirling's inequalities, yielding

$$\frac{(2k)!}{2^k \cdot k!} \geq \frac{(2k)!}{2^k \cdot \mathrm{e}\sqrt{k} \cdot (\frac{k}{\mathrm{e}})^k} \geq \frac{\sqrt{2\pi 2k} \cdot (\frac{2k}{\mathrm{e}})^{2k}}{\mathrm{e}\sqrt{k} \cdot (\frac{2k}{\mathrm{e}})^k} \geq \frac{\sqrt{4\pi}}{\mathrm{e}} \cdot \left(\frac{2k}{\mathrm{e}}\right)^k \geq \left(\frac{2k}{\mathrm{e}}\right)^k.$$

Using that $m = 2k$, we conclude that at least $\lceil (m/2)\log(m/\mathrm{e}) \rceil$ advice bits are necessary for any online algorithm to be optimal on the tree S_m. □

This lower bound is asymptotically larger by a logarithmic factor than the DPA lower bound [2], which suggests that the advice complexity of CAT increases with the degree of the tree network and not only with its size.

2.2 A Lower Bound for Competitiveness

In this section, we present a reduction from an online problem called the string guessing problem to CAT. In the *string guessing problem with unknown history* (q-SGUH), an algorithm has to guess a string of specified length z over a given alphabet of size $q \geq 2$ character by character. After guessing all characters, the algorithm is informed of the correct answer. The cost of a solution $\textsc{Alg}(I)$ is the Hamming distance between the revealed string and $\textsc{Alg}(I)$. Böckenhauer et al. [4] presented a lower bound on the number of necessary advice bits depending on the achieved fraction of correct character guesses.

We will use this lower bound for our results, reducing the q-SGUH problem to CAT by assigning each element of the alphabet to an optimal solution for a family of instances on the star tree S_d where $d = q$. The idea is to have a common prefix on all instances and the last requests specifying a unique optimal solution for the instance corresponding to the character in the string. For each character we have to guess, we insert such a star tree $S^{(i)} := S_d$ into our graph such that the graph is connected but the trees do not share edges. We can then look at each subtree independently and join the instances to an instance corresponding to the whole string. Let therefore, for some $z, d \in \mathbb{N}$, $\mathcal{T}_{z,d}$ be the set of trees that can be constructed from subtrees $S^{(i)}$ with $i \in \{1, \ldots, z\}$, such that any two subtrees share at most one vertex and the tree is fully connected. For each request of a d-SGUH instance of length z, where the optimal answer would be the string $s_1 \ldots s_z$ we now construct a sequence of requests for $S^{(i)}$ such that choosing the optimal set of requests for $S^{(i)}$ corresponds to correctly guessing s_i. Then, any algorithm that solves a fraction α of all subtrees optimally can be used to achieve a fraction α of correct guesses on the d-SGUH instance.

Theorem 2. *Every online algorithm with advice for CAT which achieves a competitive ratio of $c \leq d/(d-1)$ on any tree $T \in \mathcal{T}_{z,d}$, for $d \geq 3$ and $z, d \in \mathbb{N}$, has to read at least*

$$\left(1 - H_d\left(d - \frac{d-1}{c} - 1\right)\right) \frac{m}{d} \log d$$

advice bits, where H_d is the d-ary entropy function. □

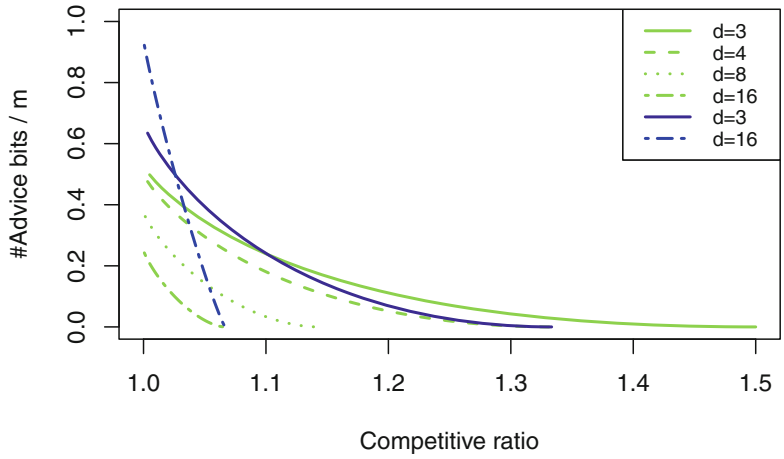

Fig. 2. Lower bound on the number of advice bits stated in Theorem 2 (light green) and Theorem 3 (dark blue) divided by the number m of edges in $T \in \mathcal{T}_{z,d}$. (Color figure online)

We can use the same tree structure to prove a better lower bound for small values of c by changing the reduction instance of CAT. This change increases the alphabet size of q-SGUH that we can reduce to instances on trees in $\mathcal{T}_{z,d}$ for $q = 2^{d-1}$, and allows to prove the following theorem.

Theorem 3. *Every online algorithm with advice for CAT which achieves a competitive ratio of* $c \leq d/(d-1\ +1/2^{d-1})$ *on any tree* $T \in \mathcal{T}_{z,d}$*, for* $d \geq 2$ *and* $z, d \in \mathbb{N}$*, has to read at least*

$$\left(1 - H_{2^{d-1}}\left(d - \frac{d}{c}\right)\right) m \cdot \frac{d-1}{d}$$

advice bits, where H_d *is the d-ary entropy function.* □

Figure 2 depicts the lower bounds of Theorems 2 and 3, respectively.

3 Upper Bounds

In the following, we present online algorithms with advice for the call admission problem on trees. In the first part, the focus will be on optimal algorithms with different advice complexities. In the second part, we discuss an algorithm whose competitiveness gradually improves with added advice bits.

3.1 Optimal Online Algorithms with Advice

The fundamental idea of the following algorithms is to encode the optimal solution using edge labels as advice. A straightforward approach is to give all requests

in $\textsc{Opt}(I)$ an identifying number and label the edges of each request with this identifier. After communicating the labels of all edges, the algorithm will be able to distinguish which request is in $\textsc{Opt}(I)$ and which is not, by checking whether all the edges of the request have the same label and no other edges have this label. As a result, the algorithm can recognize and only accept requests that an optimal solution accepts.

As for the advice complexity, we need m labels each consisting of a number in $\{0, 1, \ldots, |\textsc{Opt}(I)|\}$. Since, for any input sequence I, we have $|\textsc{Opt}(I)| \leq m$, in total $m\lceil\log(m+1)\rceil$ advice bits suffice. If $|\textsc{Opt}(I)|$ is much smaller than m, we can communicate the size of a label using a self-delimiting encoding [17], using $m\,\lceil\log(|\textsc{Opt}(I)|+1)\rceil + 2\,\lceil\log\lceil\log(|\textsc{Opt}(I)|+1)\rceil\rceil$ advice bits in total.

We will continue to use this idea of an identifier for the following algorithms in a more local manner. Instead of giving each request in $\textsc{Opt}(I)$ a global identifying number and labeling corresponding edges accordingly, we associate identifiers with an optimal request depending on the vertices incident to the request's edges.

We can picture this labeling scheme with a request as a row of dominoes, where each edge of the request represents one domino and consecutive edges have the same identifying number at their common vertex. Knowing for all edges the incident edges that are part of the same request, we can reconstruct the paths belonging to all requests in $\textsc{Opt}(I)$ and since we only need local identifiers, this reduces the size of each label.

Theorem 4. *There is an optimal online algorithm with advice for CAT that uses at most* $2m\,\lceil\log(d+1)\rceil$ *advice bits, where* d *is the maximum degree of a vertex in* T.

Proof Sketch. First, let us define such a local labeling formally. For some input sequence $I = (r_1, \ldots, r_l)$, let $\textsc{Opt}(I) \subseteq \{r_1, \ldots, r_l\}$ be an arbitrary, but fixed optimal solution for I. Furthermore, for every vertex v, let $\textsc{Opt}_v(I)$ be the subset of requests in $\textsc{Opt}(I)$ which occupy an edge incident with v. Observe that $|\textsc{Opt}_v(I)| \leq d(v)$ as there are $d(v)$ edges incident with v. For all $v \in V$, let $g_v \colon \textsc{Opt}_v(I) \to \{1, \ldots, d(v)\}$ be an injective function that assigns a number to each request in $\textsc{Opt}_v(I)$. These numbers serve as local identifiers of each request in $\textsc{Opt}_v(I)$. We define the label function lb as follows; for $e = \{v, v'\} \in E$, where $v < v'$, let

$$lb(e) := \begin{cases} (lb_v(e),\ lb_{v'}(e)) = (g_v(r),\ g_{v'}(r)) & \text{if } \exists\ r \in \textsc{Opt}(I) \text{ s.t. } e \in \text{edges}(r), \\ (0,0) & \text{otherwise.} \end{cases}$$

Thus, if an edge e is used by $r \in \textsc{Opt}(I)$, the local identifiers of r for the two vertices of e constitute the edge label; if unused, e is labeled $(0, 0)$. Observe that, if $e = \{v, v'\} \in \text{edges}(r)$ and $r \in \textsc{Opt}(I)$, it follows that $r \in \textsc{Opt}_v(I)$ and $r \in \textsc{Opt}_{v'}(I)$, so lb is well-defined. Figure 3 shows an example of a local and a global labeling side by side. Observe that, for vertex v, we have the label 4 for two edges, but no label 2. As g_v may arbitrarily assign an identifier in $\{1, \ldots, d_v\}$,

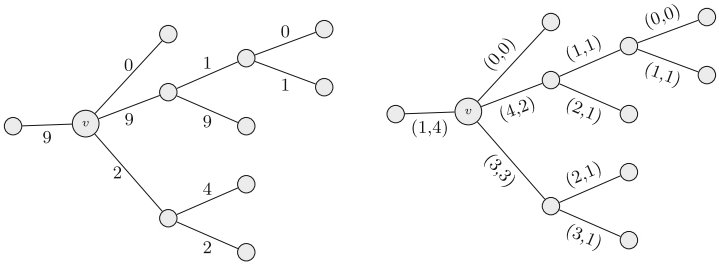

Fig. 3. Examples of a global labeling (left) and a local labeling (right) for the same optimal solution.

the assigned number to a request does not have to be minimal. In the *lb*-labeled tree T, we call $p = (v'_1, \ldots, v'_l)$ a *labeled path of length* l if p is a path in T with $v'_1 < v'_l$ and $lb_{v'_j}(\{v'_{j-1}, v'_j\}) = lb_{v'_j}(\{v'_j, v'_{j+1}\}) \neq 0$ for all $j \in \{2, \ldots, l-1\}$.

We refer to p as a *complete labeled path* if further no other edges incident to v'_1 or v'_l have label $lb_{v'_1}(\{v'_1, v'_2\})$ or $lb_{v'_l}(\{v'_{k-1}, v'_l\})$, respectively.

Consider an algorithm Alg′ that reads the labels of all edges from the advice before starting to receive any request and then computes the set P of all complete labeled paths in T. Alg′ then accepts a request $r = (v, v')$ if and only if it coincides with a complete labeled path in P.

We can prove that Alg′ accepts all requests in Opt(I), and thus is optimal, by showing that every request in Opt(I) has a coinciding path in P and that all paths in P are pairwise edge-disjoint. It remains to bound the number of advice bits used. For an edge $\{v, v'\}$, we need $\lceil \log(d(v)+1) \rceil + \lceil \log(d(v')+1) \rceil$ advice bits to communicate the label $lb(\{v, v'\})$. Hence, per vertex w, we use $d(w) \cdot \lceil \log(d(w)+1) \rceil$ advice bits. Summing up over all vertices yields the claimed bound. □

We can further improve this bound by showing that pinpointing an endvertex of a request $r \in$ Opt(I) does not require a unique identifier.

Theorem 5. *There is an optimal online algorithm with advice for CAT that uses at most* $(m-1)\lceil \log(\lfloor d/2 \rfloor + 1) \rceil$ *advice bits, where* d *is the maximum degree of a vertex in* T.

Proof Sketch. Consider a function $g_v \colon$ Opt$_v(I) \to \{0, 1, \ldots, \lfloor d(v)/2 \rfloor\}$ that assigns a non-zero identifier only to requests in Opt$_v$ that occupy two incident edges to v, otherwise it assigns identifier 0. Observe that we halve the number of identifiers needed this way.

Let us now refer to a labeled path $p = (v'_1, \ldots, v'_l)$ as complete if and only if

$$lb_{v'_1}(\{v'_1, v'_2\}) = lb_{v'_l}(\{v'_{l-1}, v'_l\}) = 0. \tag{1}$$

Again, we consider Alg′ that reads the advice $lb(e_1), \ldots, lb(e_m)$ for a tree T and computes the set P of all complete labeled paths in T according to Theorem 4.

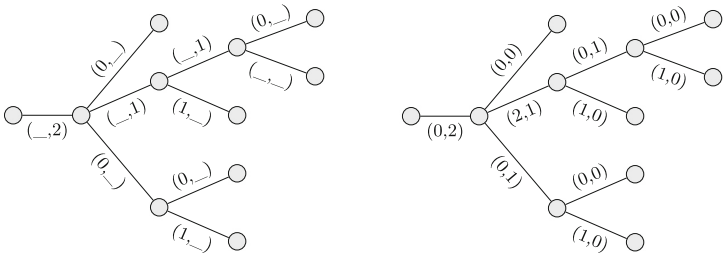

Fig. 4. Example of a labeling as used in Theorem 4 (left) and the inferred full labeling (right) for the same optimal solution.

ALG′ then accepts a request $r = (v, v')$ if and only if it coincides with a complete labeled path in P.

We can show, analogously to the proof of Theorem 4, that all paths in P are pairwise edge-disjoint and that each request in OPT(I) has a coinciding path in P (Fig. 4). Thus, as before, ALG′ accepts all requests in OPT(I) and is therefore optimal. Finally, we note that not all labels have to be communicated. Consider a vertex v and its incident edges $e'_1, \ldots, e'_{d(v)}$. Assuming that we have all labels $lb_v(e'_1), \ldots, lb_v(e'_{d(v)-1})$, we can infer the last label $lb_v(e'_{d(v)})$ as follows. If there exists only one edge $e \in \{e'_1, \ldots, e'_{d(v)}\}$ with non-zero label $lb_v(e)$, then $lb_v(e'_{d(v)}) = lb_v(e)$, since by definition of $g_v(r)$ and lb there are exactly two edges with the same non-zero label. If there is no such edge, $lb_v(e'_{d(v)}) = 0$ for the same reason. Therefore, for each vertex we only need to communicate the advice for the first $d(v) - 1$ edges and per edge only $\lceil \log(\lfloor d(v)/2 \rfloor + 1) \rceil$ advice bits. In total, ALG′ needs at most $\lceil \log(\lfloor d/2 \rfloor + 1) \rceil \cdot (m - 1)$ advice bits, where d is the maximum degree of a vertex in T. □

Note that the central idea behind the algorithms of Theorems 4 and 5 is to identify, for all inner vertices, which incident edges belong to the same request in some fixed optimal solution. We used edge labels as advice to convey this information. In what follows, we will discuss another technique to encode this information for some types of trees. First, let us examine the star tree S_d of degree d; let $I_d = (r_1, \ldots, r_{(d(d-1))/2})$ denote the instance with all possible requests in S_d of length 2.

Lemma 1. *For the instance I_d of CAT on S_d, the size of the set of solutions $\mathcal{O}(I_d)$ is at most*

$$\sum_{j=0}^{\lfloor d/2 \rfloor} \frac{d!}{2^j \cdot j! \cdot (d-2j)!}. \tag{2}$$

Proof. We construct a graph $G(I_d)$, where the vertex set corresponds to the leaf vertices of S_d. For a request r of I_d, we insert an edge in $G(I_d)$ between the respective vertices. Note that, since I_d consists of all requests between leaf vertices in S_d, we have that $G(I_d)$ is the complete graph K_d on d vertices.

Any solution $O \in \mathcal{O}(I_d)$ describes a set of edge-disjoint requests, and thus can be uniquely associated with a matching in $G(I_d)$: In the tree S_d, with requests of length 2, this is equivalent to requests having pairwise different endpoints, i.e., their corresponding edges must form a matching in $G(I_d)$. Thus, the size of the set $\mathcal{O}(I_d)$ is the number of matchings in $G(I_d)$, which is given by the Hosoya index[1] of K_d, that is, by (2). □

Now consider an arbitrary instance I^* of CAT on S_d and an optimal solution $\textsc{Opt} \in \mathcal{O}(I^*)$. Any algorithm that knows the partial solution of $\textsc{Opt}$ for requests of length 2 is optimal on I^*, as it can allocate requests of length 2, such that the edges of length-1 requests in $\textsc{Opt}$ are not blocked. Furthermore, note that this partial solution can be described by a solution in the set $\mathcal{O}(I_d)$ of instance I_d. Thus, enumerating the elements in the set $\mathcal{O}(I_d)$ and using the index of the partial solution as advice yields an algorithm that is optimal on I^*.

Corollary 1. *There exists an optimal online algorithm with advice for CAT on S_d that uses at most*

$$\left\lceil \log\left(\sum_{j=0}^{\lfloor d/2 \rfloor} \frac{d\,!}{2^j \cdot j\,! \cdot (d-2j)\,!} \right) \right\rceil \approx \left\lceil \frac{d}{2} \log\left(\frac{d}{e}\right) + \log\left(\frac{e^{\sqrt{d}}}{(4e)^{1/4}}\right) \right\rceil$$

advice bits. □

The asymptotical approximation is given by using Stirling's inequality on the bound of Lemma 1 as shown by Chowla et al. [11]. Thus, the upper bound of Corollary 1 is asymptotically of the same order as the lower bound of Theorem 1 in the previous chapter, which is constructed on a star tree S_d.

We can use the set of solutions $\mathcal{O}(I_d)$ to construct a similar algorithm as in Corollary 1 for k-ary trees of arbitrary height. The idea is to regard each inner vertex of a k-ary tree and its neighbors as a star tree with at most $k+1$ leaves. Since any k-ary tree has at most $l := (k^h - 1)/(k-1)$ inner vertices, we get subtrees $S_1, \ldots, S_l$ for which we can give advice as described before. Since the advice complexity of Theorem 5 is about twice that of Corollary 1 for a star tree S_{k+1}, this algorithm reduces the amount of advice used for each inner node, improving the upper bound for k-ary trees by a factor of about 2 when compared to Theroem 5.

Theorem 6. *There exists an optimal online algorithm with advice for CAT on k-ary trees of height h that uses at most*

$$\left\lceil \frac{k^h - 1}{k-1} \cdot \log\left(\sum_{j=0}^{\lfloor (k+1)/2 \rfloor} \frac{(k+1)\,!}{2^j \cdot j\,! \cdot (k+1-2j)\,!} \right) \right\rceil$$

advice bits. □

[1] The Hosoya index, or Z-index, describes the total number of matchings in a graph. Note that it counts the empty set as a matching. The above expression for the Hosoya index of K_d is given by Tichy and Wagner [19].

3.2 Competitiveness and Advice

A popular approach to create competitive algorithms for online problems is to divide the requests into classes, and then randomly select a class. Within this class, requests are accepted greedily and requests of other classes are dismissed.

Awerbuch et al. [1] describe a version of the "classify and randomly select" algorithm for the CAT problem based on vertex separators as follows. Consider a tree with n vertices. There has to exist a vertex v_1' whose removal results in disconnected subtrees with at most $n/2$ vertices. Iteratively choose, in each new subtree created after the $(i-1)$-th round, a new vertex to remove and add it to the set V_i; vertices in this set are called level-i vertex separators. This creates disjoint vertex classes $V_1, V_2, \ldots, V_{\lceil \log n \rceil}$. We can now separate incoming requests into levels. A request r is a level-i_V request if $i_V = \min_j(V_j \cap V(r) \neq \emptyset)$ where $V(r)$ is the set of vertices in the path of the request. The algorithm then chooses a level i_V^* uniformly at random and accepts any level-i_V^* request greedily, i.e., an incoming level-i_V^* request is accepted if it does not conflict with previously accepted requests.

This randomized algorithm is $2\lceil \log_2 n \rceil$-competitive in expectation and can be easily adapted to the advice model by choosing the accepted class using advice. When we reduce the number of classes by a factor of $1/p$ for some $p \in \{1, \ldots, \lceil \log n \rceil\}$, the number of advice bits necessary to communicate the level index will decrease, but we can expect the greedy scheduling to perform worse.

Theorem 7. *For any $p \in \{1, \ldots, \lceil \log n \rceil\}$ there is an online algorithm with advice for CAT that uses $\lceil \log \log n - \log p \rceil$ advice bits and is*

$$\left((2^{p+1} - 2) \cdot \left\lceil \frac{\log n}{p} \right\rceil \right) \text{-competitive.}$$

Proof Sketch. We define the set $\mathrm{Join}(i_V, p)$ to include all requests in I of levels $i_V, \ldots, \min\{i_V + (p-1), \log n\}$. We say that request r is in a subtree S if all its edges are in S, and use "block" in the sense of two requests having at least one edge in common. Observe that requests of level i_V or higher have all edges in a subtree created by removing vertices in $V_1, \ldots, V_{i_V - 1}$. We call such a subtree a level-i_V subtree. Let $\textsc{Opt}(I)$ be an optimal solution to I; it can be proven by induction on p that for all $i_V \in \{1, \ldots, \lceil \log n \rceil\}$, any request r in a subtree of level i_V can block at most $2^{p+1} - 2$ other requests in $\textsc{Opt}(I) \cap \mathrm{Join}(i_V, p)$. We can conclude that, for a fixed $p \in \{1, \ldots, \lceil \log n \rceil\}$, any greedy scheduler is $(2^{p+1} - 2)$-competitive when requests are restricted to a level $i_{V'}$, for some $i_{V'} \in \{1, \ldots, \lceil (\log n)/p \rceil\}$. This follows directly from the induction hypothesis. The competitive ratio and advice complexity are easily deduced from there on. □

Corollary 2. *There exists a $2\lceil \log n \rceil$-competitive algorithm for CAT that uses $\lceil \log \log n \rceil$ advice bits.* □

References

1. Awerbuch, B., Bartal, Y., Fiat, A., Rosén, A.: Competitive non-preemptive call control. In: Proceedings of SODA 1994, pp. 312–320. SIAM (1994)
2. Barhum, K., et al.: On the power of advice and randomization for the disjoint path allocation problem. In: Geffert, V., Preneel, B., Rovan, B., Štuller, J., Tjoa, A.M. (eds.) SOFSEM 2014. LNCS, vol. 8327, pp. 89–101. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-04298-5_9
3. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
4. Böckenhauer, H.-J., Hromkovič, J., Komm, D., Krug, S., Smula, J., Sprock, A.: The string guessing problem as a method to prove lower bounds on the advice complexity. Theoret. Comput. Sci. **554**, 95–108 (2014)
5. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R.: On the advice complexity of the k-server problem. J. Comput. Syst. Sci. **86**, 159–170 (2017)
6. 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). https://doi.org/10.1007/978-3-642-10631-6_35
7. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R., Mömke, T.: Online algorithms with advice: the tape model. Inf. Comput. **254**, 59–83 (2017)
8. Böckenhauer, H.-J., Komm, D., Wegner, R.: Call admission problems on grids with advice (extended abstract). In: Epstein, L., Erlebach, T. (eds.) WAOA 2018. LNCS, vol. 11312, pp. 118–133. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-04693-4_8
9. Böckenhauer, H.-J., Hromkovič, J., Komm, D., Královič, R., Rossmanith, P.: On the power of randomness versus advice in online computation. In: Bordihn, H., Kutrib, M., Truthe, B. (eds.) Languages Alive. LNCS, vol. 7300, pp. 30–43. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31644-9_2
10. Burjons, E., Frei, F., Smula, J., Wehner, D.: Length-weighted disjoint path allocation. In: Böckenhauer, H.-J., Komm, D., Unger, W. (eds.) Adventures Between Lower Bounds and Higher Altitudes. LNCS, vol. 11011, pp. 231–256. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98355-4_14
11. Chowla, S., Herstein, I.N., Moore, W.K.: On recursions connected with symmetric groups I. Can. J. Math. **3**, 328–334 (1951)
12. Dobrev, S., Královič, R., Pardubská, D.: Measuring the problem-relevant information in input. Theoret. Inform. Appl. (RAIRO) **43**(3), 585–613 (2009)
13. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. Theoret. Comput. Sci. **412**(24), 2642–2656 (2011)
14. Gebauer, H., Komm, D., Královič, R., Královič, R., Smula, J.: Disjoint path allocation with sublinear advice. In: Xu, D., Du, D., Du, D. (eds.) COCOON 2015. LNCS, vol. 9198, pp. 417–429. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21398-9_33
15. 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). https://doi.org/10.1007/978-3-642-15155-2_3
16. Komm, D.: Advice and Randomization in Online Computation. Ph.D. thesis, ETH Zurich (2012)
17. Komm, D.: An Introduction to Online Computation - Determinism, Randomization Advice. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-319-42749-2

18. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM **28**(2), 202–208 (1985)
19. Tichy, R.F., Wagner, S.: Extremal problems for topological indices in combinatorial chemistry. J. Comput. Biol. **12**(7), 1004–1013 (2005)