Treasure Hunt with Advice*

Dennis Komm[1], Rastislav Královič[2], Richard Královič[3], and Jasmin Smula[1]

[1] Department of Computer Science, ETH Zürich, Switzerland
{dennis.komm,jasmin.smula}@inf.ethz.ch
[2] Department of Computer Science, Comenius University, Bratislava, Slovakia
kralovic@dcs.fmph.uniba.sk
[3] Google Zurich, Inc., Switzerland
ri.kralovic@gmail.com

Abstract. The node searching problem (a.k.a. treasure hunt) is a fundamental task performed by mobile agents in a network and can be viewed as an online version of the shortest path problem: an agent starts in a vertex of an unknown weighted undirected graph, and its goal is to reach a given vertex. The cost is the overall distance (measured by the weights of the traversed edges) traversed by the agent. We consider the setting in which the agent sees the identifier of the vertex it is located in, the weights of the incident edges, and also the identifiers of the neighboring vertices. We analyze the problem from the point of view of advice complexity: at the beginning, the agent has a tape with an advice string that gives some a priori information about the input instance. This information has no restricted form; instead, the aim is to study the relationship between the size of this advice and the competitive ratio that can be obtained. We give tight bounds of the form $\Theta(n/r)$ bits of advice for a competitive ratio r (possibly depending on the number of vertices n). In particular, this means that an a priori knowledge of any graph parameter (which would be of size $O(\log n)$) cannot yield a competitive ratio better than $\Omega(n/\log n)$. Moreover, we give a lower bound on the expected competitive ratio of any randomized online algorithm for treasure hunt.

1 Introduction

Problems that involve traversing a graph in a certain way belong to the fundamentals of graph theory [11], and a vast amount of effort has been invested into their study. In such problems, there is an entity (an *agent*) that starts in some vertex of a given graph, and its goal is to explore the graph in some manner such as by traversing all vertices/edges (possibly under some restrictions), finding a (shortest) path to a given vertex, etc. The oldest and most extensive treatment has been the offline case, where the graph and the starting vertex are given as an input and the task is to find the optimal exploration route. The online versions, where the graph is unknown and the agent can only observe its immediate surroundings, have also received significant attention over the last decades [2,18].

* Partially funded by the SNF grant 200021-146372 and grant VEGA 1/0979/12.

C. Scheideler (Ed.): SIROCCO 2015, LNCS 9439, pp. 328–341, 2015.
DOI: 10.1007/978-3-319-25258-2_23

In this paper, we consider the following problem, which is called the *treasure hunt* problem. Given is an n-vertex undirected graph with non-negative edge weights. We assume the vertices to have unique identifiers. An agent starts in some vertex, and its goal is to move to a given vertex (the *treasure*); the identity of the treasure vertex is revealed once the agent enters it. The agent can freely move from a vertex to any neighboring vertex, incurring a cost equal to the weight of the traversed edge (an edge can be traversed an arbitrary number of times, each time incurring the same cost). When located at a vertex, the agent can see the identifier of the current vertex, the identifiers of all neighbors, and the weights of the incident edges. This model is known as the *fixed graph scenario* and was introduced by Kalyanasundaram and Pruhs [21]. The agent has (polynomial) memory and can perform an arbitrary computation, but it cannot modify the graph in any way; e. g., it cannot mark any vertices. Our aim is to construct a deterministic algorithm for the agent that minimizes the worst-case competitive ratio (i. e., the ratio of the cost incurred by the algorithm and the optimal cost). First, we observe that a simple greedy agent is $O(n)$-competitive, and that no deterministic agent can achieve a better competitive ratio (even in the unweighted case, where all edges have weight 1). Next, we analyze the problem using the framework of advice complexity.

The advice complexity has been introduced in the context of online problems [4,9,10,20] and distributed problems [16,17]. In both settings, the algorithm has to cope with the lack of information about the instance it is working on (the future input requests, the network topology, etc.).

There are many results that study the impact of some a priori information about the instance on the performance of the algorithm. While the traditional approach is qualitative and analyzes information of a certain type (e. g., the agent may know the size or diameter of the network), the advice complexity approach is quantitative: the agent obtains, at the beginning of the computation, a binary string (the *advice*) that describes the current instance. There is no restriction on the type of the information; it can be viewed as being prepared by a computationally unbounded oracle that knows the algorithm of the agent and the whole input instance (i. e., the network topology, the edge weights, the starting vertex of the agent, and the location of the treasure). The main incentive is to study the relationship between the size of the advice string and the competitive ratio that can be obtained. In recent years, the advice complexity of a large number of online problems was studied; examples include paging [4], the k-server problem [5, 10, 19, 28], and the knapsack problem [6]. Furthermore, it is possible to use special reduction methods to transfer hardness results on the advice complexity from one problem to another [3, 7, 10].

We show that without any advice, any algorithm is $\Omega(n)$-competitive, even on unweighted graphs, and even using randomization. On the other hand, n bits of advice are sufficient to obtain an optimal solution. Next, we consider unweighted graphs (where all edges have weight 1). We show an asymptotically tight trade-off between the size of the advice and the competitive ratio: for any (not necessarily constant) r, there is an algorithm that uses $O(n/r)$ bits of advice

and that achieves a competitive ratio r. On the other hand, any deterministic algorithm that achieves a competitive ratio r needs $\Omega(n/r)$ bits of advice.

Related Work

In 1977, Rosenkrantz et al. [27] proposed the following problem: an agent is initially located in a vertex of an unknown weighted undirected graph, and is able to move from a vertex to any neighboring vertex, inducing a cost equal to the weight of the traversed edge. When located at a vertex, the agent can see the identifier of that vertex, the weights of incident edges, and also the identifiers of its neighbors. Its goal is to traverse all vertices of the graph and induce as little cost as possible. The performance is evaluated in terms of the competitive ratio, i. e., the worst case ratio of the cost induced by the algorithm and the optimum cost. It has been shown that the natural greedy nearest neighbor algorithm is $\Theta(\log_2 n)$-competitive, and a question was posed whether there is any algorithm with a constant competitive ratio. Despite much effort, the question is still open for the general case (although many results are known for special graph classes), the best general lower bound being $5/2 - \varepsilon$ [8]. In the case of directed graphs, the problem is much better understood: the competitive ratio of any (even randomized) algorithm is at least $\Omega(n)$ and $O(n)$-competitiveness is indeed achievable [15].

The same model has been used to study the treasure hunt problem in directed graphs [15]. It has been shown that, on strongly connected weighted graphs, no deterministic or randomized algorithm can have a bounded competitive ratio. For strongly connected unweighted graphs, the best competitive ratio for deterministic algorithms is $\Theta(n^2)$; for randomized algorithms, a lower bound of $\Omega(n)$ is known.

The treasure hunt problem has been studied in unweighted undirected graphs in the framework of advice complexity by Miller and Pelc [24], in the model in which the agent does not see the identifiers of the neighboring vertices (i. e., it only sees the identifier of the vertex it is located in). In particular, in our model, the agent can identify a neighboring vertex as already visited, which can be exploited by the algorithm. The performance of the algorithm in the model of Miller and Pelc was measured in terms of the number of edges e, the competitive ratio r, and the cost of the optimal solution D. The lower and upper bounds $\Omega\big(D \log_2(\frac{e}{rD})\big)$ and $O\big(D \log_2(\frac{e}{r})\big)$ have been obtained, respectively.

A similar problem of finding a shortest path with a given source and sink was studied for a specific class of graphs by Papadimitriou and Yannakakis [26] and Fiat et al. [14]. The model used in these papers differs from the one studied here in the way in which the vertices are revealed to the agent.

Another related problem is the *rendezvous* problem, where two agents want to meet in an unknown graph [24, 25, 29]. Note that treasure hunt is a special case of rendezvous where the position of one of the agents is fixed.

Finally, note that such problems belong to the class of *search games*, which are further distinguished according to different parameters. Other famous problems from this class include the linear search problem (also known as the cow path problem) [1, 22] and the ANTS (ants nearby treasure search) problem [12, 13].

2 Our Contribution

Let $G = (V, E)$ with $|V| = n$ be an undirected graph with non-negative edge weights $\omega(e)$, $e \in E$ ($\omega(u, v)$, $u, v \in V$, respectively). Let u_0 be the starting vertex of the agent, and let $u_0, u_1, \ldots, u_\ell$ be the shortest path to the treasure vertex u_ℓ. Since the cost of the optimum may be zero, we use the so called non-strict competitive ratio with an additive constant to even out the effects of the special cases with zero (or near-zero) optimum: an algorithm is said to be r-competitive, if there is a constant α such that on any input instance the algorithm incurs cost at most $r \cdot OPT + \alpha$, where OPT is the cost of the optimal solution (i. e., the length of the shortest $u_0 - u_\ell$ path).

As a first observation, let us consider the competitive ratio of a deterministic algorithm without any advice.

Theorem 1. *There is a deterministic algorithm without advice for the treasure hunt problem in weighted graphs that achieves a competitive ratio of $O(n)$.*

Proof. Consider the following simple greedy algorithm on a given graph $G = (V, E)$. The agent works in rounds and maintains a set of vertices that have been visited so far. Let S_i be the set of vertices visited after i rounds, with $S_0 = \{u_0\}$ containing only the starting vertex. In each round i, the agent starts from the vertex u_0, and a new vertex is explored as follows. Let v_i be the vertex from $V \setminus S_{i-1}$ with the shortest distance from u_0. (Note that, since all vertices from S_{i-1} have been already visited, the agent knows the costs of all the outgoing edges; moreover, since also the identifier of the destination is known for an edge, the agent can distinguish which edges lead to vertices in $V \setminus S_{i-1}$.) The agent traverses the shortest path to v_i, which reveals the neighborhood of v_i, and returns back to u_0; the new set is $S_i := S_{i-1} \cup \{v_i\}$. The exploration ends whenever S_i contains the treasure vertex u_ℓ.

Obviously, there are at most n rounds. In each round i, the treasure vertex is located in $V \setminus S_{i-1}$, so the treasure is at least as far as v_i. Hence, in each round, the agent traverses at most twice the optimal cost. □

The previous theorem is essentially tight, since even when using randomization, and even on unweighted graphs, an asymptotically better competitive ratio cannot be achieved.

Theorem 2. *Any randomized algorithm for the treasure hunt problem on unweighted stars has competitive ratio $\Omega(n)$.*

Proof. Using Yao's principle [30], the expected cost of any randomized algorithm on a worst-case instance from a set of instances $\mathcal{I}$ is at least the expectation of the cost of the best deterministic algorithm over any probability distribution on the instances. Let us take for $\mathcal{I}$ all the instances where the graph is a star with $n-1$ leaves, the agent starts in the root, and the treasure is in some leaf. Consider any deterministic algorithm, and a probability distribution where every leaf is chosen to be the treasure vertex uniformly at random. The algorithm visits the

leaves in a fixed order, until it arrives to the leaf with the treasure. Hence, for any i with $1 \leq i \leq n-1$, the probability that the agent finds the treasure in step $2i-1$ is $1/(n-1)$. Therefore, the expected cost is

$$\frac{1}{n-1} \sum_{i=1}^{n-1} 2i - 1 \in \Omega(n)$$

as claimed. □

On the other hand, linear advice is sufficient to achieve optimality, as can be seen from the following theorem.

Theorem 3. *There is an optimal deterministic algorithm with n bits of advice for the treasure hunt problem in weighted graphs.*

Proof. In order to be optimal, the agent must traverse the shortest path $u_0, u_1, \ldots, u_\ell$. The advice is used to query the status of the vertices, i. e., whether they belong to this optimal path or not. Let the agent be located at a vertex u_i. First, it queries, one bit per vertex, the status of all neighbors that have not been queried yet, in the order of increasing identifiers. Since the agent is deterministic, the advice string can be prepared based on the input instance in the correct order. After this step, the agent knows which neighbors of u_i are on the optimal path, and in particular it knows the set of neighbors P that are of the form u_j with $j > i$. Obviously, $u_{i+1} \in P$. Moreover, the weight $\omega(u_i, u_{i+1})$ is the smallest of all weights $\omega(u_i, u_j)$, $u_j \in P$, since otherwise $u_0, u_1, \ldots, u_\ell$ would not be shortest. Lastly, there may be several vertices u_j in P with the same weight $\omega(u_i, u_j) = \omega(u_i, u_{i+1})$. An arbitrary vertex among them can be chosen (deterministically) since in this case all the edges on the shortest path up to u_j must have zero weight. □

Again, Theorem 3 cannot be asymptotically improved: advice of linear size is needed even to obtain a constant competitive ratio on unweighted graphs due to Theorem 5.

For the rest of the paper, let us focus on unweighted graphs, i. e., graphs where all edges have weight 1. As our main result, we show an asymptotically tight trade-off between the size of the advice and the best possible competitive ratio. We start by stating an upper bound. To this end, however, we need the following lemma. The lemma basically deals with special vertex separators of size 1 and has been mentioned as folklore by Lipton and Tarjan [23]. We provide the proof for the sake of completeness.

Lemma 1. *Let $G = (V, E)$ be a connected graph with $|V| = n > 6$ vertices. Then there are two sets $C, D \subseteq V$ such that $C \cup D = V$, $|C \cap D| = 1$, both $|C| > n/3$, $|D| > n/3$, and each of them induces a connected subgraph.*

Proof. It is sufficient to prove the lemma for trees, since then it can be applied to a spanning tree of an arbitrary connected graph. Let $G = (V, E)$ be an arbitrary

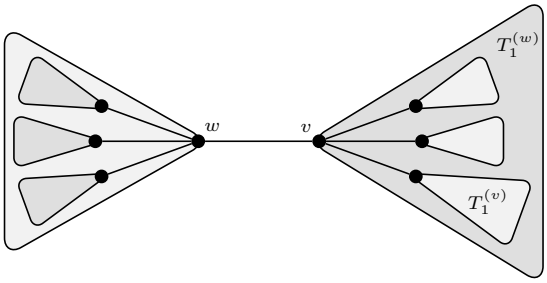


Fig. 1. Schematic drawing of the graph G in the situation described in Lemma 1. The tree $T_1^{(w)}$ contains more than $n/2$ vertices, so the subtree rooted at w must contain less than $n/2$ vertices. The subtree $T_1^{(v)}$ that contains more than $n/2$ vertices must be one of the subtrees within $T_1^{(w)}$.

tree. For any vertex w, let G decompose into n_w trees $T_1^{(w)}, \ldots, T_{n_w}^{(w)}$ when removing w from V. For every $w \in V$ and every i with $1 \leq i \leq n_w$, denote by $V_i^{(w)}$ the vertex set of $T_i^{(w)}$.

First we prove that there is a vertex $w \in V$ such that, for each i with $1 \leq i \leq n_w$, we have $|V_i^{(w)}| \leq n/2$. Let us assume by contradiction that for each vertex $w \in V$ there exists some index j such that $|V_j^{(w)}| > n/2$. Since for any vertex w there cannot be more than one such subtree, without loss of generality, let $T_1^{(w)}$ be the unique subtree from vertex w with $|V_1^{(w)}| > n/2$. Now consider a vertex w such that $|V_1^{(w)}|$ is minimal among all vertex sets $V_1^{(u)}$ for all $u \in V$, and let v be the (unambiguous) neighbor of w in $T_1^{(w)}$ (see Fig. 1).

From the point of view of v, the subtree $T_i^{(v)}$ that is rooted in w has less than $n/2$ vertices and thus cannot be the subtree $T_1^{(v)}$ with $|V_1^{(v)}| > n/2$. Hence, the subtree of v with more than $n/2$ vertices must be one of the subtrees contained in $T_1^{(w)}$, and therefore $|V_1^{(v)}| < |V_1^{(w)}|$. This is a contradiction to the minimality of $|V_1^{(w)}|$.

Now consider a vertex $w \in V$ such that each $|V_i^{(w)}| \leq n/2$ for all $1 \leq i \leq n_w$. If there is a tree $T_j^{(w)}$ that has $|V_j^{(w)}| \geq n/3$ vertices, then let

$$C \coloneqq T_j^{(w)} \cup \{w\}$$

and

$$D \coloneqq (V \setminus C) \cup \{w\}.$$

Else, if all the $T_i^{(w)}$ have $|V_i^{(w)}| < n/3$ vertices, assign the (vertices of) the subtrees $T_i^{(w)}$ one by one greedily to the set C or D that currently contains fewer vertices. When all vertices of all these subtrees are assigned, we additionally add w to both C and D. Hence, in the end, the cardinalities of the two parts differ by at most $n/3$, which means that both sets contain at least

$$\frac{n}{2} - \frac{n/3}{2} + 1 \geq \frac{n}{3} + 1$$

vertices. □

Now we use the result from Lemma 1 to prove an upper bound on the number of advice bits sufficient to obtain a competitive ratio of r.

Theorem 4. *Let $r := r(n)$ be any function of n such that $18 < r < n$ and r is divisible by 9. There is an r-competitive algorithm for the treasure hunt problem on unweighted graphs that uses $O(n/r)$ bits of advice.*

Proof. Recall that the agent starts in vertex u_0, and that the treasure is located in u_ℓ, so the optimal cost is ℓ. The agent works its way towards u_ℓ in *rounds* that consist of traversing a number of edges, and possibly reading some advice bits. In order to keep track of the advice spent and the distance traversed, two types of accountings are used for the purpose of the analysis: at the beginning, each vertex has a *charge* $9/r$, and the agent has *credit* 0. At some point in time, we may decide to *harvest* some vertices, adding their charge to the agent's credit. We shall make sure that no vertex is harvested twice, and the agent reads only as many advice bits as is its credit. This way, the overall size of the advice is bounded by $9n/r$. The second type of accounting keeps track of the traveled distance: every move of the agent is *booked* to some edge (u_i, u_{i+1}) from the optimal path. In order to bound the competitive ratio, we assert that no edge is booked more than r times.

During each round i, the agent maintains three disjoint sets of vertices. The set H_i contains all vertices that have already been harvested in previous rounds; for each vertex $v \in H_i$, either v or its neighbor has been visited in the previous rounds. The set T_i contains all vertices that have already been visited but not yet harvested. Finally, the set B_i, the *boundary vertices*, are those vertices from $V \setminus (H_i \cup T_i)$ that have a neighbor in T_i. At the beginning, we have $T_1 := \{u_0\}$, $H_1 := \emptyset$, and B_1 contains all the neighbors of u_0.

There are two different kinds of rounds: *traversal* rounds and *advice* rounds. If, in round i, the size of the set $T_i \cup B_i$ is at most $r/3$, the agent traverses these vertices, and books the cost of the traversal to one edge of the optimal path. On the other hand, if $T_i \cup B_i$ is too big for the traversal cost to be amortized, the agent reads one advice bit to narrow down the set of vertices that must be traversed. The rounds are grouped into phases such that each phase starts with a number (possibly zero) of advice rounds and ends with one traversal round.

Let us consider a phase p consisting of rounds $h+1, \ldots, h+m$. We ensure the following invariants hold at the beginning of each round i.

(a) B_i contains some vertex u_j from the shortest path such that no vertices u_k, $k > j$ are in H_i. Let j^* be the maximum index such that $u_{j^*} \in B_i$; we call u_{j^*} the *distinguished vertex* and $e_i := (u_{j^*}, u_{j^*+1})$ the *distinguished edge* (neither the distinguished vertex, nor the distinguished edge is known to the agent, they are for the purpose of the analysis only).

(b) No costs have been booked to any edge (u_k, u_{k+1}) for $j^* \leq k \leq \ell - 1$.
(c) The agent is located at some vertex $v \in T_{h+1}$.
(d) $T_i \cup B_i$ is connected.

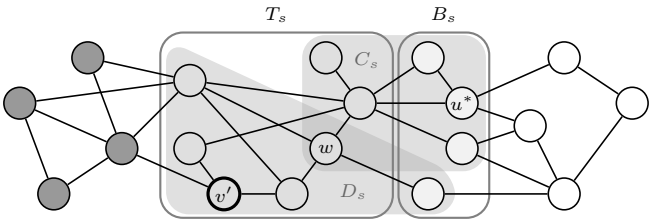
Now let us describe how the algorithm works in greater detail (refer to Fig. 2). The computation starts with round 1 of phase 1. For the sake of simplicity, we say that the preceding (dummy) round was round 0. At the beginning of its computation, the agent is located at u_0. The initial values for the sets are, as we have already mentioned above, $T_1 := \{u_0\}$, $H_1 := \emptyset$, and $B_1 := \big\{v \in V \mid (u_0, v) \in E\big\}$. The distinguished vertex is the last vertex from $u_1, \ldots, u_\ell$ that is a neighbor of u_0. It is easy to verify that all invariants hold.

If, in round i, it holds that $|T_i \cup B_i| \geq r/3$, the agent executes an advice round: it internally splits $T_i \cup B_i$ into two parts C_i and D_i using Lemma 1. Then it reads one bit of advice indicating which one of the sets C_i and D_i contains the distinguished vertex u^*. Without loss of generality, let this be C_i. Note that u^* might even be contained in both C_i and D_i, since these sets intersect in one vertex w. If this is the case, the oracle specifies the set C_i to be the one containing u^*.

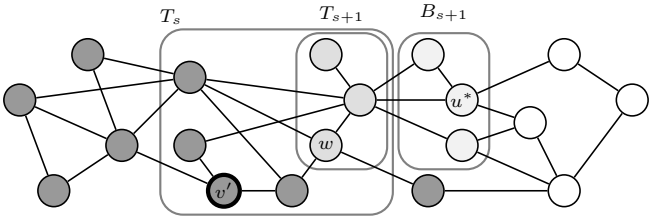
Then the vertices from $D_i \setminus \{w\}$ are harvested to pay for the advice bit that it just read. Since $D_i \subseteq T_i \cup B_i$, D_i and H_i are disjoint. Hence, the vertices in $D_i \setminus \{w\}$ have not been harvested yet, and Lemma 1 guarantees that both C_i and D_i contain at least $r/9+1$ vertices. Thus, the agent gains enough credit by harvesting the vertices from $D_i \setminus \{w\}$ to pay for one advice bit. It sets $H_{i+1} := H_i \cup D_i \setminus \{w\}$, $T_{i+1} := T_i \cap C_i \subseteq T_i$, and $B_{i+1} := C_i \setminus T_i = B_i \cap C_i$. This implies that $u^* \in B_{i+1}$, and the distinguished edge does not change. Invariant (a) is trivially fulfilled. As no costs were booked to any edges in this round, also invariant (b) remains true. Invariant (c) holds since the agent did not move at all in this round. To verify that invariant (d) holds, note that $T_{i+1} \cup B_{i+1} = (T_i \cup B_i) \cap C_i = C_i$, which is connected by Lemma 1.

If, on the other hand, $|T_i \cup B_i| < r/3$, the agent executes a traversal round. Hence, this is the last round (round $h + m$) of phase p. Due to invariant (c) the agent is at some vertex $v \in T_{h+1}$. Before the agent starts to traverse $T_{h+m} \cup B_{h+m}$, it must first enter this set, which incurs cost at most $r/3$ that is booked onto the distinguished edge (which exists due to invariant (a)). The limit of $r/3$ is implied by the fact that T_{h+1} was generated by the traversal round of the previous phase and the properties of traversal rounds shown below.

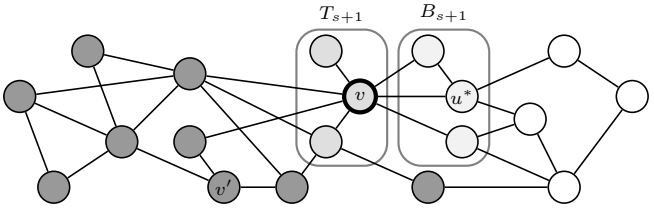
Now, the agent uses a depth-first search to traverse $T_{h+m} \cup B_{h+m}$, and as soon as it comes across the destination vertex u_ℓ, the algorithm terminates. Otherwise, the traversal incurs cost of less than $2r/3$ that are booked to e_{h+m}. The agent sets $T_{h+m+1} := T_{h+m} \cup B_{h+m}$, since all vertices contained in these sets have been traversed now but have not been harvested yet, and $H_{h+m+1} := H_{h+m}$, as no vertices have been harvested in this round. The agent also computes B_{h+m+1} according to T_{h+m+1} and H_{h+m+1}. From invariant (a) we can conclude that one endpoint of the present distinguished edge e_{h+m} must be $u_{j^*} \in B_{h+m}$ and the other one $u_{j^*+1} \notin (H_{h+m+1} \cup T_{h+m+1})$. Thus, $u_{j^*+1} \in B_{h+m+1}$ is a vertex that guarantees that invariant (a) is fulfilled. However, u_{j^*+1} is not necessarily the distinguished vertex of the next round; this is the last vertex u_k that is



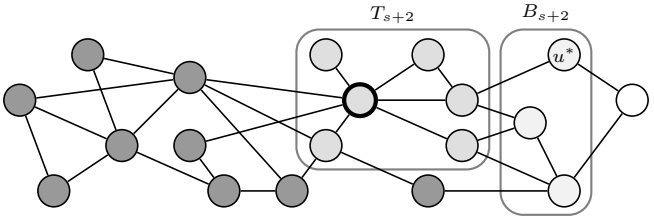
(a). In the preceding round, $s-1$, the agent traversed the set T_s and ended in some vertex $v' \in T_s$. Now, in round s, the set $T_s \cup B_s$ is too large to be traversed, so the agent splits it into two parts C_s and D_s, which overlap in w.



(b). After round s, the agent harvested all vertices from $D_s \setminus \{w\}$ and updated the sets accordingly. The set $T_{s+1} \cup B_{s+1}$ is small enough to be traversed, but the agent is located at the vertex $v' \in T_s \setminus T_{s+1}$, and before starting the traversal, it must first move to some vertex in T_{s+1}.



(c). When round $s+1$ begins, the agent has moved to $v \in T_{s+1}$ and can now start its traversal.



(d). After round $s+1$, all vertices from $T_{s+1} \cup B_{s+1} = T_{s+2}$ have been traversed. The agent updates the sets T_{s+2} and B_{s+2} and the distinguished vertex u^* accordingly.

Fig. 2. An example of a sequence of rounds performed by the agent.

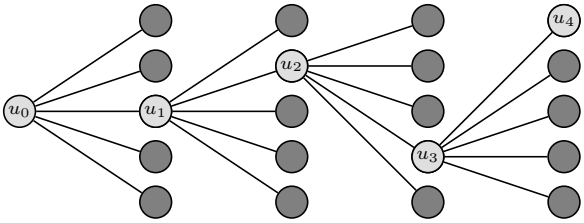


Fig. 3. Example of a pq-tree for $p = 5$, $q = 4$, and $n = 21$. The starting vertex u_0 is the only vertex on level 0. There are q additional levels and p vertices per level. There is one designated vertex per level, colored in gray; u_i is the designated vertex on level i. Each such vertex u_i for $0 \leq i \leq q-1$ is connected to all vertices on level $i+1$. The vertex u_q is the destination vertex.

in B_{h+m+1}, for some $k \geq j^* + 1$. Then, the new distinguished edge $e_{h+m+1} = (u_k, u_{k+1})$ is the edge where the optimal path leaves B_{h+m+1} for the last time. Thus, invariants (a) and (b) remain true. At the end of round $h+m$, which is also the beginning of phase $p+1$, the agent is located at some vertex $v \in T_{h+m} = T_{h+m+1}$, making sure that also invariant (c) holds. Invariant (d) holds as well since T_{h+m+1} is connected due this invariant from the previous round and each vertex of B_{h+m+1} is connected to a vertex of T_{h+m+1}.

The traversal stops, whenever the agent enters the treasure vertex u_ℓ for the first time. For every i with $0 \leq i \leq \ell-1$, every edge $\{u_i, u_{i+1}\}$ is the distinguished edge of at most one phase, and thus cost of at most r are booked to it: not more than $r/3$ for the adjustment move before the traversal of the current search space and at most $2r/3$ for the traversal itself. Hence, the total cost of the agent, adding the additional cost for the last move, is at most $r \cdot \ell + 1$, whereas the cost of an optimal solution is ℓ, and thus the algorithm is r-competitive. Furthermore, each vertex is harvested at most once, amounting for the advice complexity $9n/r$. □

The algorithm from the previous theorem cannot be asymptotically improved, as we show in the next theorem. In the proof we shall use a class of instances called pq-trees (see Fig. 3), which are defined as follows: for given p, q, n, such that $n > pq$, consider q columns (called levels), each containing p vertices. On each level, there is one designated vertex, and every designated vertex on level $1 \leq i < q$ is connected to all vertices of level $i+1$. Additionally, there is a single vertex on level 0 (root) that is connected to all vertices on level 1, and $\gamma := n - pq - 1$ dummy vertices connected to the root. This way the vertices form a tree with n vertices. The identifiers of the vertices are arbitrary but fixed for a given set of parameters p, q, n. The starting vertex is the root, and the treasure vertex is the designated vertex on level q. There are p^q possible ways to choose the designated vertices $u_1, \ldots, u_q$, and thus $\mathcal{I}$ contains $|\mathcal{I}| = p^q$ different instances, each of them with optimal solution q. We prove the following theorem.

Theorem 5. *Let $r := r(n)$ be any function of n, such that $1 \leq r < n/18$ for each n. Any algorithm for the treasure hunt problem on pq-trees with competitive ratio r needs at least $\Omega(n/r)$ bits of advice.*

Proof. Consider any treasure hunt algorithm $\mathcal{A}$ for pq-trees, with competitive ratio r. This means that there is a constant α, such that on any pq-tree with n vertices, the cost of the algorithm is at most $r \cdot OPT + \alpha$. Let us fix some (large enough) n. We construct a particular pq-tree with n vertices, where the algorithm reads more than $\beta\frac{n}{r}$ bits of advice for some constant $\beta > 0$ that depends on the algorithm and α, but neither on the function r nor on n.

Set $q \coloneqq \lfloor n/kr \rfloor$ for a suitable constant k specified later, and choose p such that $n = pq + \gamma$ for some $1 \leq \gamma \leq q$. Consider all p^q instances $\mathcal{I}$ with parameters p, q, n. The agent traverses the graph, starting at u_0, until it finally reaches the destination vertex u_q. Until it does, it must visit at least one vertex on each level. The order in which $\mathcal{A}$ traverses the vertices on each level i is fixed for any fixed instance. On any level i with $1 \leq i \leq q$, let a_i be such that $\mathcal{A}$ visits u_i as the a_i-th vertex on level i. Let us assume without loss of generality that $\mathcal{A}$ never visits a leaf twice and never returns to level $i-1$ once it has found u_i. We can identify each instance $I \in \mathcal{I}$ with the characteristic vector $(a_1, a_2, \ldots, a_q)$, with $a_i \in \{1, \ldots, p\}$ for $1 \leq i \leq q$, and the property that the cost of $\mathcal{A}$ on instance I is at least $\sum_{i=1}^{q} (2a_i - 1)$ (we ignore the potential dummy vertices).

Let us call an instance I *good* if $\mathcal{A}$ achieves a competitive ratio of at most r on it, i. e., if the cost of $\mathcal{A}$ on I is at most $qr + \alpha$, which implies

$$\sum_{i=1}^{q} a_i \leq \frac{q(r + 1 + \alpha/q)}{2} .$$

For convenience, let us denote $d \coloneqq (r + 1 + \alpha/q)/2$. This implies that an instance is good if, for its corresponding characteristic vector $(a_1, \ldots, a_q)$,

$$\sum_{i=1}^{q} a_i \leq qd . \tag{1}$$

For the following argumentation, we interpret each algorithm that uses b bits of advice as a set $\{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_{2^b}\}$ of deterministic algorithms, as is often done with online algorithms with advice. Since $\mathcal{A}$ reads at most b advice bits and is r-competitive, there is at least one deterministic algorithm $\mathcal{A}_j$ that computes a solution with a competitive ratio of at most r on at least $p^q/2^b$ instances. From now on, let us consider this particular deterministic algorithm $\mathcal{A}_j$, and let us define the set of good instances for $\mathcal{A}_j$ to be $\mathcal{I}^+$. Thus we have

$$|\mathcal{I}^+| \geq \frac{p^q}{2^b} . \tag{2}$$

Now let us bound $|\mathcal{I}^+|$, the number of good instances for $\mathcal{A}_j$, from above. For any good instance, the corresponding characteristic vector must contain at least $q/2$ entries a_i with value at most $2d$; hence, the number of good instances is upper-bounded by the number of vectors $(a_1, \ldots, a_q)$, where $a_i \in \{1, \ldots, p\}$, with at least $q/2$ entries with value at most $2d$. To bound this term from above, we make the following considerations. The number of vectors of length $q/2$ with values of at most $2d$ is $(2d)^{q/2}$, the number of vectors of length $q/2$ with values between 1

and p is $p^{q/2}$. The number of possibilities to join two vectors of these two different kinds to construct a vector of length q is $\binom{q}{q/2}$. The same vector of length q might be generated by joining different pairs of vectors of length $q/2$. Nevertheless, these considerations yield an upper bound. The number of characteristic vectors with at least $q/2$ entries with value at most $2d$, and thus also the number of good instances, is therefore

$$|\mathcal{I}^+| \leq (2d)^{\frac{q}{2}} \cdot p^{\frac{q}{2}} \cdot \binom{q}{\frac{q}{2}} . \tag{3}$$

Putting (2) and (3) together yields

$$\frac{p^q}{2^b} \leq (8dp)^{\frac{q}{2}} .$$

We rearrange this inequality to solve it for b and obtain

$$b \geq \frac{q}{2} \cdot \log_2\Big(\frac{p}{8d}\Big) .$$

If we show that $p/8d > 2$, then

$$b \geq \frac{q}{2} \geq \frac{1}{2}\Big(\frac{n}{kr} - 1\Big) \geq \frac{1}{2k}\frac{n}{r} - \frac{1}{2} .$$

Since $n/r > 18$, choosing $\beta \coloneqq 1/(2k) - 1/36$ yields

$$\frac{1}{2k}\frac{n}{r} - \frac{1}{2} > \beta \cdot \frac{n}{r} .$$

It must hold that $\beta > 0$, i. e., $k < 18$. It remains to show that $p/8d > 2$. Recall that $p = (n-\gamma)/q \geq (n-q)/q$. Substituting d, one gets

$$\frac{p}{8d} \geq \frac{n-q}{4q(r+1)+4\alpha} ,$$

which is more than 2 if $q < (n-8\alpha)/(8r+9)$. Finally, we show that for a suitable choice of k we can obtain

$$q = \Big\lfloor \frac{n}{kr} \Big\rfloor \leq \frac{n}{kr} < \frac{n-8\alpha}{8r+9} .$$

In order for the last inequality to hold, we need to choose k such that

$$k > \frac{n}{n-8\alpha}\frac{8r+9}{r} .$$

The second term is always at most 17, and the first term converges to 1 with increasing n. Hence, there is a large enough n (depending on α) such that $k = 17.5$ is suitable. □

3 Conclusion

We analyzed the treasure hunt problem on undirected graphs in the framework of advice complexity. In particular, we have shown a tight bound of $\Theta(n/r)$ on advice for algorithms with competitive ratio r on unweighted graphs. A natural next step would be to extend this result to weighted graphs.

Furthermore, the upper bound from Theorem 3 can be improved. An anonymous reviewer pointed out a proof that uses a slightly more involved argument and allows to decrease the number of advice bits to $2/3n$.

Acknowledgement. The authors would like to thank Hans-Joachim Böckenhauer and Juraj Hromkovič for very valuable discussions, and an anonymous reviewer.

References

1. Baeza-Yates, R.A., Culberson, J.C., Rawlins, G.J.E.: Searching in the plane. Information and Computation 106(2), 234–252 (1993)
2. Berman, P.: On-line searching and navigation. In: Fiat, A., Woeginger, G.J. (eds.) Online Algorithms 1996. LNCS, vol. 1442, pp. 232–241. Springer, Heidelberg (1998)
3. 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. Theoretical Computer Science 554, 95–108 (2014)
4. 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)
5. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R.: On the advice complexity of the k-server problem. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 207–218. Springer, Heidelberg (2011)
6. 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)
7. Boyar, J., Favrholdt, L.M., Kudahl, C., Mikkelsen, J.W.: Advice complexity for a class of online problems. In: Mayr, E.W., Ollinger, N. (eds.) Proc. of the 32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015). LIPIcs, vol. 30, pp. 116–129. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2015)
8. Dobrev, S., Královič, R., Markou, E.: Online graph exploration with advice. In: Even, G., Halldórsson, M.M. (eds.) SIROCCO 2012. LNCS, vol. 7355, pp. 267–278. Springer, Heidelberg (2012)
9. Dobrev, S., Královič, R., Pardubská, D.: Measuring the problem-relevant information in input. RAIRO ITA 43(3), 585–613 (2009)
10. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. Theoretical Computer Science 412(24), 2642–2656 (2011)
11. Euler, L.: Solutio problematis ad geometriam situs pertinentis. Commentarii Academiae Scientiarum Imperialis Petropolitanae 8, 128–140 (1736)
12. Feinerman, O., Korman, A.: Memory lower bounds for randomized collaborative search and implications for biology. In: Aguilera, M.K. (ed.) DISC 2012. LNCS, vol. 7611, pp. 61–75. Springer, Heidelberg (2012)

13. Feinerman, O., Korman, A., Lotker, Z., Sereni, J.S.: Collaborative search on the plane without communication. In: Kowalski, D., Panconesi, A. (eds.) Proc. of the 31st ACM Symposium on Principles of Distributed Computing (PODC 2012), pp. 77–86 (2012)
14. Fiat, A., Foster, D.P., Karloff, H.J., Rabani, Y., Ravid, Y., Vishwanathan, S.: Competitive algorithms for layered graph traversal. SIAM Journal on Computing 28(2), 447–462 (1998)
15. Förster, K.-T., Wattenhofer, R.: Directed graph exploration. In: Baldoni, R., Flocchini, P., Binoy, R. (eds.) OPODIS 2012. LNCS, vol. 7702, pp. 151–165. Springer, Heidelberg (2012)
16. Fraigniaud, P., Ilcinkas, D., Pelc, A.: Oracle size: A new measure of difficulty for communication tasks. In: Proc. of the 25th Annual ACM symposium on Principles of distributed computing (PODC 2006), pp. 179–187. ACM (2006)
17. Fraigniaud, P., Ilcinkas, D., Pelc, A.: Communication algorithms with advice. Journal of Computer and System Sciences 76(3–4), 222–232 (2010)
18. Ghosh, S.K., Klein, R.: Online algorithms for searching and exploration in the plane. Computer Science Review 4(4), 189–201 (2010)
19. Gupta, S., Kamali, S., López-Ortiz, A.: On advice complexity of the k-server problem under sparse metrics. In: Moscibroda, T., Rescigno, A.A. (eds.) SIROCCO 2013. LNCS, vol. 8179, pp. 55–67. Springer, Heidelberg (2013)
20. 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)
21. Kalyanasundaram, B., Pruhs, K.R.: Constructing competitive tours from local information. Theoretical Computer Science 130(1), 125–138 (1994)
22. Kao, M.-Y., Reif, J.H., Tate, S.R.: Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. Information and Computation 131(1), 63–79 (1996)
23. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM Journal on Applied Mathematics 36(2), 177–189 (1979)
24. Miller, A., Pelc, A.: Tradeoffs between cost and information for rendezvous and treasure hunt. In: Aguilera, M.K., Querzoni, L., Shapiro, M. (eds.) OPODIS 2014. LNCS, vol. 8878, pp. 263–276. Springer, Heidelberg (2014)
25. Miller, A., Pelc, A.: Fast rendezvous with advice. In: Gao, J., Efrat, A., Fekete, S.P., Zhang, Y. (eds.) ALGOSENSORS 2014, LNCS 8847. LNCS, vol. 8847, pp. 75–87. Springer, Heidelberg (2015)
26. Papadimitriou, C.H., Yannakakis, M.: Shortest paths without a map. Theoretical Computer Science 84(1), 127–150 (1991)
27. Rosenkrantz, D.J., Stearns, R.E., Lewis II, P.M.: An analysis of several heuristics for the traveling salesman problem. SIAM Journal on Computing 6(3), 563–581 (1977)
28. Renault, M.P., Rosén, A.: On online algorithms with advice for the k-server problem. In: Solis-Oba, R., Persiano, G. (eds.) WAOA 2011. LNCS, vol. 7164, pp. 198–210. Springer, Heidelberg (2012)
29. Ta-Shma, A., Zwick, U.: Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences. ACM Transactions on Algorithms 10(3), 12:1–12:15 (2012)
30. Yao, A.C.-C.: Probabilistic computations: Toward a unified measure of complexity. In: Proc. of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977), pp. 222–227 (1977)