Theoretical Computer Science 463 (2012) 62–72

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Online graph exploration: New results on old and new algorithms☆

Nicole Megow[a], Kurt Mehlhorn[a], Pascal Schweitzer[b,*]

[a] Max Planck Institute for Informatics, Campus E1 4, D-66123 Saarbrücken, Germany

[b] Australian National University, Building 108, North Road, Canberra, ACT 0200, Australia

ARTICLE INFO

Keywords:
Graph exploration
Online algorithms
Competitive analysis

ABSTRACT

We study the problem of exploring an unknown undirected connected graph. Beginning in some start vertex, a searcher must visit each node of the graph by traversing edges. Upon visiting a vertex for the first time, the searcher learns all incident edges and their respective traversal costs. The goal is to find a tour of minimum total cost. Kalyanasundaram and Pruhs (Constructing competitive tours from local information, Theoretical Computer Science 130, pp. 125–138, 1994) proposed a sophisticated generalization of a Depth First Search that is 16-competitive on planar graphs. While the algorithm is feasible on arbitrary graphs, the question whether it has constant competitive ratio in general has remained open. Our main result is an involved lower bound construction that answers this question negatively. On the positive side, we prove that the algorithm has constant competitive ratio on any class of graphs with bounded genus. Furthermore, we provide a constant competitive algorithm for general graphs with a bounded number of distinct weights.

1. Introduction

In an exploration problem an agent, or searcher, has to construct a complete map of an environment without any *a priori* knowledge of its topology. The searcher makes all its decisions based on partial local knowledge and gathers new information on its exploration tour. Exploration problems appear in various contexts, such as robot motion planning in hazardous or inaccessible terrain, maintaining security of large networks, and searching, indexing, and analyzing digital data on the internet [39,7,25].

We study the online graph exploration problem on undirected connected graphs $G = (V, E)$. We assume that the vertices are labeled so that the searcher is able to distinguish them. Each edge $e = (u, v) \in E$ has a non-negative real weight $|e|$, also called the length or the cost of the edge. Beginning in a distinguished start vertex $s \in V$, the searcher learns G according to the following online paradigm, also known as *fixed graph scenario* [30]: whenever the searcher visits a vertex, it learns all incident edges, their weights, and the labels of their end vertices. To explore a new vertex, the searcher traverses previously learned edges in the graph. For traversing an edge, the searcher has to pay the respective edge cost. The task is to find a tour that visits all vertices V and returns to the start vertex. The goal is to find a tour of minimum total length. An illustration of this model (see [30]) is the scenario where vertices correspond to cities and upon arrival in a city the searcher sees the road signs of routes to other cities including distance information.

A standard technique to measure the quality of online algorithms is *competitive analysis* [10], which compares the outcome of an algorithm with an *optimal offline solution*. For our graph exploration problem, the corresponding offline

☆ An extended abstract of this work appeared in Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP), Springer, 2011.

* Corresponding author. Tel.: +61 261259665.
E-mail addresses: nmegow@mpi-inf.mpg.de (N. Megow), mehlhorn@mpi-inf.mpg.de (K. Mehlhorn), pascal@mpi-inf.mpg.de, pascal.schweitzer@anu.edu.au (P. Schweitzer).

0304-3975/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2012.06.034

problem is the fundamental *Traveling Salesman Problem* (TSP), one of the most studied optimization problems which is in general even NP-hard to approximate [34,27]. It asks for a shortest tour that visits every vertex of a graph known in advance. For a positive number c, we call an online exploration algorithm *c-competitive*, if it computes for any instance a tour of total length at most c times the optimal offline tour through all vertices. The *competitive ratio* of an algorithm is the infimum over all c such that the algorithm is c-competitive.

For general graphs with arbitrary weights no constant competitive algorithm is known. A promising candidate was introduced by Kalyanasundaram and Pruhs [30]. Their algorithm ShortCut is a sophisticated generalization of DFS obtained by introducing a parameterized *blocking condition* that determines when to diverge from DFS. They prove an upper bound of 16 on its competitive ratio in planar graphs. The algorithm itself is defined for general graphs. However, since its introduction almost two decades ago, it has been open if ShortCut has constant competitive ratio in general. There has been no progress on this question since then, and in fact, all subsequent results concerned with our graph exploration setting only apply to simple cycles: Asahiro et al. [2] showed that NN yields a competitive ratio of $3/2$ on simple cycles. Additionally, a lower bound of $5/4$ for any deterministic online algorithm is proven. Both lower and upper bounds were improved by Miyazaki et al. [35]. They give a more sophisticated algorithm which takes additionally the current total tour length into account. They prove that, on simple cycles, this algorithm achieves the best possible competitive ratio of $1 + \sqrt{3}$. It is not clear how the algorithm can be generalized and applied to more complex graphs.

In the special case that all edges have equal weight, a standard *Depth First Search* (DFS) is 2-competitive. It yields a total tour not larger than twice the size of a minimum spanning tree (*MST*), a lower bound on the optimal tour. This is optimal in the unit-weight case as Miyazaki et al. [35] showed.

In the traditional offline environment, the simple and fast greedy algorithm *Nearest Neighbor* (NN) has been studied intensively. It repeatedly chooses the next vertex to be visited as an unexplored vertex closest to the current location. Note that in the online scenario, even though only partial information on the input graph is available, the nearest neighbor can always be identified. Moreover, a shortest path to the nearest neighbor can be determined online. Thus, the worst case ratio for the greedy algorithm of $\Theta(\log n)$ shown by Rosenkrantz et al. [40] also applies to our online scenario. This worst case ratio is tight even on planar unit-weight graphs, which follows from a nice and simple lower bound construction of graphical instances by Hurkens and Woeginger [28].

1.1. Our contribution

We revisit algorithm ShortCut proposed by Kalyanasundaram and Pruhs [30]. We elaborate on the sophistication of the underlying idea, but report also a precarious issue in the given formal implementation. We propose a reformulation, which we call Blocking, highlighting the elaborate idea from [30], and adapt the proof of [30] to assure that the reformulation has constant competitive ratio for planar graphs. Here, a concise observation allows us to simplify the proof and to generalize it to graphs of bounded genus. More precisely, we generalize the upper bound on the competitive ratio of 16 for planar graphs to a bound of $16(1 + 2g)$ for graphs of genus at most g.

As our main result we show that Blocking does not have a constant competitive ratio on general graphs. We use a classical construction of Erdős [17] of graphs with large girth and large minimum degree to construct complex graphs for which Blocking has arbitrarily large competitive ratio. Considering the fact that we have shown that Blocking is constant competitive for classes of graphs that have bounded genus, it seems plausible that similarly heavy machinery is indeed necessary for the lower bound construction. Interestingly, the lower bound construction requires only two different edge weights. For this case, we show a constant competitive algorithm. More generally, we provide in this paper an online algorithm that is $2k$-competitive on general graphs with at most k distinct weights. Our algorithm generalizes DFS to an algorithm that hierarchically performs depth first searches on subgraphs induced by restricted weights. For arbitrary graphs with arbitrary weights we round weights up to the nearest power of 2 and apply the same algorithm. With this modification the algorithm has a competitive ratio of $\Theta(\log n)$ in general.

1.2. Related work

Exploration and search problems in unknown environments have been studied extensively and there is an immense body of literature on them; see the surveys [7,39,26] and [31, Chap. 1]. Such problems have been studied already in the 1960s, mainly from a game-theoretic perspective with one player searching for another player that is hiding; see, e.g., the book by Gal [24]. More recent research on online motion planning, aiming explicitly for worst-case performance guarantees on the total travel distance, was initiated by Baeza-Yates et al. [5].

Exploration problems are closely related to search problems. In search problems, the searcher must find a shortest path to an object (of any type, e.g., point, infinite line, etc.) in an unknown location. Exploration problems however, ask for a walk through an unknown environment such that any point of the environment is "visible" from some point of the walk. Details on the respective definition of visible depend on the particular specification of the environment and the searcher. While the solution in a search problem is compared to the offline shortest path from the origin to the object's location, an exploration tour is compared to an optimal tour after which the searcher has seen the entire environment. Generally, the geometry of the search environment can be arbitrary — a bounded or unbounded space, with or without obstacles, two, three, or higher

dimensional. However, in many particular applications, it is possible to abstract from the geometry of the real environment and model the unknown search space as a graph, in which the searcher may only move along edges.

Some of the first formal models for exploring an unknown graph were proposed by Papadimitriou and Yannakakis [38] in the context of finding a shortest path between two given points. Deng and Papadimitriou [11] initiated research on fully exploring a graph (drawing a map). In contrast to our problem, they consider the task of exploring all *edges* in a directed labeled unknown graph (with unit-weight edges). In this model, the searcher is given at any time the number of unexplored edges leaving the vertex, but not their endpoints. Notice that the corresponding offline problem is the polynomially solvable Chinese postman problem [16], in contrast to the TSP in our setting. In this variant the deficiency of a graph [32], a parameter measuring how far the graph is from being Eulerian, plays a crucial role. This exploration problem has been studied extensively in directed [1,21,20,33] and undirected graphs [37,12]. Numerous variants were considered, e.g., routing multiple searchers [6,22,15], models that impose additional constraints on the searcher, such as being tethered, i.e., attached to the start vertex by a rope or cable of particular length [13], or having a tank of limited capacity which forces the searcher to return periodically to the start for refueling [4,8], and exploration problems in graphs without unique labeling but with some other additional information; see [25,23] and the literature therein.

Our problem of exploring all *vertices* of a labeled undirected graph is in some sense also a variant of the initial problem in [11]. In particular, on trees the problem of exploring all vertices is equivalent to exploring all edges. Apart from the aforementioned previous work on our problem in planar graphs [30] and cycles [35,2], it has been studied on un-weighted trees also for multiple synchronously moving searchers [15,14,22]. Here the objective is to minimize the total length of the longest tour among all searchers. In this setting, the communication capability crucially influences the performance of such collective exploration.

Even though the online graph exploration problem considered in this paper has the classical TSP as the corresponding offline problem, another class of online problems is typically regarded as *online TSP* in the literature. Ausiello et al. [3] introduced a model in which the graph is given in advance and the vertices to be visited appear online over time. This means that new vertices appear as the salesman proceeds independently of his current position, which is in strong contrast to our model. The corresponding offline problem is a TSP with release dates. The online problem has been studied extensively in the past decade for various cost functions and numerous generalizations such as dial-a-ride problems, bounded capacities, multiple servers, and asymmetric variants. We refer the interested reader to the literature overviews in [9,29].

2. The exploration algorithm of Kalyanasundaram and Pruhs

We now discuss the algorithm ShortCut, that was proposed and analyzed by Kalyanasundaram and Pruhs [30].

Definition 1. A vertex is *explored* once it has been visited by the searcher. An edge is *explored* once both endpoints are explored. A *boundary edge* (u, v) is an edge with an explored end vertex u and an unexplored end vertex v.

We adopt the convention that for a boundary edge, the first entry is always the vertex that has already been explored. For a set of edges E' we let $|E'| = \sum_{e \in E'} |e|$.

Algorithm ShortCut can be seen as a sophisticated variant of DFS. The crucial ingredient is a *blocking condition* depending on a fixed parameter $\delta > 0$, which determines when to diverge from DFS.

Definition 2. At any point in time during the exploration of the graph, a boundary edge $e = (u, v)$ is said to be *blocked*, if there is a boundary edge $e' = (u', v')$ with u' explored and v' unexplored which is shorter than e (i.e., $|e'| < |e|$) and for which the length of any shortest known path from u to v' is at most $(1 + \delta) \cdot |e|$.

Intuitively, the exploration algorithm ShortCut performs a standard DFS but traverses a boundary edge only if it is not blocked. Suppose the searcher is at a vertex u and considers traversing a boundary edge (u, v). If (u, v) is blocked then its traversal is postponed, possibly forever; otherwise the searcher traverses (u, v). Traversing (u, v) and exploring v may cause another edge (x, y), whose traversal was delayed earlier, to become unblocked. Then the shortest path from v to y is added as a virtual edge (called a jump edge in [30]) to the graph and can be traversed virtually like any real edge.

It is important to carefully update the blocking-state of edges as the algorithm proceeds. In particular, an edge which has become unblocked, after having previously been blocked, may become blocked again. This may be the case if a new shorter path from an unblocked edge to another boundary edge is revealed. We argue in Example 1 that disregarding reblocking will cause an unbounded worst case ratio, even for planar graphs.

Example 1 (*Reblocking Example*)**.** Consider the planar graph depicted in Fig. 1. W.l.o.g. we move first to vertex u_w and follow the chain of unit-weight edges until we reach z. Now, all heavy edges become blocked by the edge (v_w, a). Notice that none of the boundary edges incident to the start vertex block any heavy edge, since $(1 + \delta)w < (1 + \delta)w + 3$. We move back to explore (v_w, a), which unblocks all heavy edges. (The algorithm ShortCut would add the jump edges (a, v_i), $1 \le i \le w - 1$ to the graph.) One of the heavy edges, say (z, v_1), is explored by first moving to z and then traversing (z, v_1). (In ShortCut this corresponds to traversing a jump edge.) Moving further to u_1, a cycle of explored and boundary edges is closed and a new shortest path from the unexplored endpoints u_i, $2 \le i \le w$ to z (length $w + 3$) is revealed. These paths to z are short enough such that the unit-weight boundary edges now block the heavy edges. This reblocking is crucial since otherwise all heavy edges will be traversed which leads to a worst case ratio growing at least linearly in w.

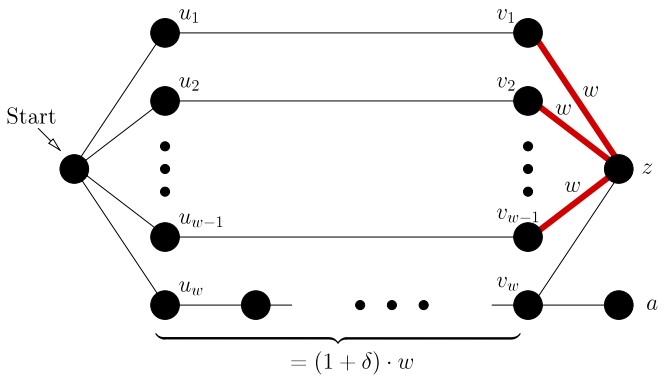

Fig. 1. Reblocking example. Given $\delta > 0$, choose a weight $w > \max\{1, 3/\delta\}$ such that $(1+\delta)\cdot w$ is integral. All edges have unit-weight except the heavy (red) edges with weight w.

Algorithm 1 The exploration algorithm $\mathsf{Blocking}_\delta(G, y)$

Input: A partially explored graph G, and a vertex y of G that is explored for the first time.

1: **while** there is an unblocked boundary edge $e = (u, v)$, with u explored and v unexplored, such that $u = y$ or such that e had previously been blocked by some edge (u', y) **do**
2: walk a shortest known path from y to u
3: traverse $e = (u, v)$
4: $\mathsf{Blocking}_\delta(G, v)$
5: walk a shortest known path from v to y
6: **end while**

The example shows that we require means of reblocking edges. This important issue is not explicitly addressed in the algorithm description in [30]. In particular, no means of blocking or removing jump edges after their creation are provided therein. A closer study of the behavior of ShortCut on the example shows that the analysis of [30] implicitly assumes that at the point in time when a jump edge is traversed, the corresponding original edge is not blocked.

In Algorithm 1, we formalize our interpretation of the algorithmic idea by Kalyanasundaram and Pruhs. To distinguish it from [30] and since the (parameterized) blocking condition is a very subtle and key ingredient, we choose the name $\mathsf{Blocking}_\delta$. To explore the entire graph starting in vertex s, we call Algorithm 1 as $\mathsf{Blocking}_\delta(G_s, s)$, where G_s is the partially explored graph in which only s has been visited so far.

Remark 1. In our exploration scenario, the online algorithm only has access to and knowledge of the explored vertices and the boundary vertices of a partially explored graph and the edges induced by these vertices. However, to avoid having to define partially explored graphs more formally, we assume that the algorithm interacts with the input graph G, and the known data gets updated whenever new vertices are explored.

We now give a proof of the fact that $\mathsf{Blocking}_\delta$ has constant competitive ratio on planar graphs. Note that the algorithm itself does not require planarity. Our proof differs from the one in [30] in using an additional argument which allows an easier handling of the recurrence.

Theorem 1. *Algorithm* $\mathsf{Blocking}_\delta$ *is* $2(2+\delta)(1+2/\delta)$*-competitive on planar graphs.*

Proof. Since in every iteration of the while loop a new vertex is explored, the algorithm terminates. We first argue that all vertices are eventually explored. Suppose otherwise.

Let $e = (u, v)$ be a shortest boundary edge upon termination of the algorithm. Being a shortest boundary edge, it cannot be blocked at the point of termination. However, it must have been blocked at some point, as otherwise it would have been explored when u was explored. Thus, there is a last point in time at which e becomes unblocked due to the exploration of some edge (x, y). Observe that by reaching y only edges having y as an endpoint become explored. But this contradicts the fact that the call $\mathsf{Blocking}_\delta(G, y)$ does not trigger the exploration of v.

We let P be the set of edges that are traversed during some execution of Line 3.

For each iteration of the while loop, we charge all costs that occur in the execution of Lines 2, 3 and 5 to the edge in P traversed in Line 3. Since any execution of Line 3 explores a new vertex, every edge is charged in at most one while loop iteration.

The cost charged to any edge e is at most $2(2+\delta)|e|$. Indeed, either the edge had previously not been blocked, in which case the cost is simply $2|e|$, or the edge e had previously been blocked by some edge ending in y, and therefore (by the definition of blocking) the distance from y to the starting point of e is at most $(1+\delta)\cdot|e|$. Thus Lines 2, 3 and 5 provoke costs of at most $(1+\delta)\cdot|e|$, $|e|$, and $(2+\delta)\cdot|e|$, respectively.

Let MST be a minimum spanning tree that shares a maximum number of edges with P. It suffices now to show that $|P| \leq (1 + 2/\delta)|MST|$ in order to get an overall cost of at most $2(1 + \delta)(1 + 2/\delta)|MST|$.

Claim 1. *If an edge $e \in P \setminus MST$ is contained in a cycle C in $P \cup MST$, then the cycle C has length at least $(2 + \delta)|e|$.*

Proof. Suppose otherwise. On the cycle C, consider the edge $e' = (u, v) \in P \setminus MST$ with $|e'| \geq |e|$ that is charged the latest. W.l.o.g. suppose e' is traversed from u to v at the time it is charged. Due to the choice of MST, the edge e' is strictly larger than any edge in $C \cap MST$. Otherwise we could replace e' with an edge in MST to obtain a smaller minimum spanning tree or to obtain a minimum spanning tree that shares more edges with P. At the time e' is charged, e' is a boundary edge, and therefore not the whole cycle has been explored. Thus there is a boundary edge different from e' on the cycle. Moreover at this point in time e' is not blocked. On the cycle C there are two paths from u to v, namely the edge e' and the path $C - e'$. Let e'' be the first boundary edge encountered when traversing $C - e'$ starting from u towards v. Since we assume the cycle has length less than $(2 + \delta)|e| \leq (2 + \delta)|e'|$ and e' is not blocked, we conclude that e'' is not smaller than e' and therefore not in MST. This yields a contradiction to the fact that e' is the edge in $P \setminus MST$ with $|e'| \geq |e|$ that is charged the latest and shows the claim. □

(Note that, since we have not used the planarity of the graph, Claim 1 is true for any graph.)

Consider a planar embedding of the graph induced by the edges in $P \cup MST$. We iteratively define for every edge $e \in P \setminus MST$ a cycle C_e in the following way. In each step we choose an edge that, together with edges in MST and edges to which a cycle has already been assigned, closes a face cycle. Note that for two distinct edges $e, e' \in P \setminus MST$ the associated cycles C_e and $C_{e'}$ are different.

Every edge in $MST \cup P$ is contained in at most two such cycles, since they form a set of distinct face cycles. For an edge in $P \setminus MST$ one of these cycles is C_e. In fact these cycles are exactly all face boundaries apart from the boundary of the outer face.

Thus we get that

$$|P \setminus MST| \leq \frac{1}{1+\delta} \sum_{e \in P \setminus MST} |C_e - e| \leq \frac{1}{1+\delta}(2|MST| + |P \setminus MST|),$$

and therefore $|P \setminus MST| \leq (2/\delta)|MST|$.

Overall we conclude $|P| \leq |P \setminus MST| + |P \cap MST| \leq (2/\delta + 1)|MST|$. □

Choosing $\delta = 2$ minimizes the term $2(2 + \delta)(1 + 2/\delta)$ in the theorem.

Corollary 1. *Algorithm* $\mathsf{Blocking}_2$ *is* 16*-competitive on planar graphs.*

3. Graphs of bounded genus

The proof of Theorem 1 can be generalized to show that $\mathsf{Blocking}_\delta$ is an exploration algorithm with constant competitive ratio on graphs of bounded genus. The genus of a graph is the least genus of an orientable closed surface on which the graph can be embedded. Recall that the genus of the sphere is 0, while the genus of a torus is 1, the genus of a double torus is 2, and so on.

We recall some basic definitions for graphs embedded on surfaces. For a thorough introduction to graphs on surfaces we refer to [36]. A graph embedded on a surface partitions the surface into vertices, edges and faces. The Euler characteristic $\chi(S)$ of a closed surface S is the number of faces plus the number of vertices minus the number of edges in a triangulation of the surface. It is a basic fact that all triangulations yield the same value. The genus of a connected orientable surface is the maximum number of disjoint closed simple curves, cutting along which does not disconnect the surface. The Euler characteristic of a closed orientable surface of genus g is $2 - 2g$. We now extend the result from the previous section to graphs of genus at most g.

Theorem 2. *Algorithm* $\mathsf{Blocking}_\delta$ *is* $2(2 + \delta)(1 + 2/\delta)(1 + 2g)$*-competitive on graphs of genus at most g.*

Proof. To show this we define the set P of charged edges as in the proof of Theorem 1. Recall that each edge in P is charged with a cost of at most $2(2 + \delta)|e|$. Let again MST be a minimal spanning tree that intersects a maximum number of edges of P. Consider an embedding of the input graph on an orientable closed surface of genus g. We choose $MST' \subseteq MST \cup P$ to be a maximal superset of MST obtained by repeatedly adding edges that do not separate two faces, i.e., are incident with only one face. (Topologically this can also be viewed as adding a set of non-separating cycles, after contracting MST to a single point.) Since the addition of a non-separating edge increases the Euler characteristic of the surface bounded by the edges, and a surface of genus g has Euler characteristic $2 - 2g$, there are at most $2g$ edges in $MST' \setminus MST$.

In case MST' does not bound a topological disk, we artificially add non-separating edges each of weight $|MST|$ to the graph induced by P, to obtain a superset MST'' of MST' that bounds a topological disk. These edges are artificial in the sense that they do not need to be edges of G. By the Euler characteristic argument above, there are at most $2g$ edges in $MST'' \setminus MST$ in total.

All edges in P, and thus, all edges in $MST'' \setminus MST$ have a weight not larger than $|MST|$, since otherwise they would be blocked, until the whole minimum spanning tree has been explored. This implies $|MST''| \leq |MST| + 2g|MST|$.

We adapt the claim of the planar case:

Claim 2. *If an edge $e \in P \setminus MST''$ is contained in a cycle C in $P \cup MST''$, then the cycle C has length at least $(2+\delta)|e|$.*

Proof. To show the claim we distinguish two cases. If, on the one hand, C does not contain an edge in $MST'' \setminus MST'$ then C is a cycle in the original graph and the claim follows as in the proof of Theorem 1, since planarity was not used to show Claim 1. If, on the other hand, C contains an edge from $MST'' \setminus MST'$ then C has length at least $|MST| + |e|$. Since MST is a minimum spanning tree, the edge e is also contained in a different cycle C' that runs entirely in $MST \cup \{e\}$. From the previous case it now follows that $(2+\delta)|e| \leq |C'| \leq |C|$, which proves the claim. □

Since MST'' bounds a disk, as before, we iteratively define for every edge $e \in P \setminus MST''$ a cycle C_e in the following way. In each step we choose an edge that together with edges in MST'' and edges to which a cycle has already been assigned closes a face cycle. As in the proof of Theorem 1, the cycles C_e and $C_{e'}$ are different, if e and e' are different. Furthermore, every edge in $P \cup MST''$ is contained in at most two cycles, and for edges in $P \setminus MST''$ one of these cycles is C_e. Therefore,

$$|P \setminus MST''| \leq \frac{1}{1+\delta} \sum_{e \in P \setminus MST''} |C_e - e| \leq \frac{1}{1+\delta}(2|MST''| + |P \setminus MST''|),$$

and thus $|P \setminus (MST'')| \leq (2/\delta)|MST''|$.

Finally, we conclude that $|P| \leq (1+2/\delta)|MST''| \leq (1+2/\delta)(1+2g)|MST|$. Overall, with the same arguments as in the proof of Theorem 1, we conclude that $\mathsf{Blocking}_\delta$ is $2(2+\delta)(1+2/\delta)(1+2g)$-competitive on graphs of genus g. □

Choosing $\delta = 2$ optimizes the upper bound.

Corollary 2. *Algorithm* $\mathsf{Blocking}_2$ *is* $16(1+2g)$*-competitive on graphs of genus at most* g*.*

4. A lower bound construction

We now show that there is no $\delta > 0$ for which Algorithm $\mathsf{Blocking}_\delta$ has constant competitive ratio on arbitrary graphs. To this end, we construct graphs on which the competitive ratio of the algorithm is unbounded. The graphs only contain edges of weight 1 and edges of weight $w > 1$, called *heavy* edges.

Overview of the construction. Our construction relies on a *base graph* H which is a bipartite graph with large girth and large degree. Each vertex of the base graph is replaced by a gadget containing edges of negligible weight. More precisely, the vertices in the one bipartition class are replaced by a so-called release gadget, while the vertices in the other bipartition class are replaced by a so-called collection gadget. For each edge e of the base graph there is a heavy weight edge in the final construction connecting the gadgets corresponding to the endpoints of e. The order in which the vertices are traversed is controlled by the interaction between the gadgets, eventually forcing the algorithm to traverse all heavy weight edges. In the construction, the high girth of the base graph prevents unwanted blocking of edges, while the high degree ensures that the number of heavy edges is large, leading to excessive cost for the algorithm.

We now describe the construction in detail. To show that $\mathsf{Blocking}_\delta$ does not have constant competitive ratio, it suffices to use base graphs with arbitrarily large girth and arbitrarily large minimum degree. However, to obtain an explicit lower bound for the competitive ratio of $\mathsf{Blocking}_\delta$ we need graphs with specific bounds on girth and degree. The existence of these graphs is guaranteed by the following lemma, which extends a classical construction of Erdős [17].

Lemma 1. *For all $\bar{d} \in \mathbb{N}$ and $\delta \in \mathbb{R}^+$ there exists a connected bipartite graph H that has $\mathcal{O}(\bar{d}^{\delta+2})$ vertices with minimum degree at least $\bar{d}$, maximum degree at most $2\bar{d}$, and a girth of $g \geq \delta + 2$.*

Proof. For $\bar{d} \leq 1$ the statement is trivial. For $\bar{d} \geq 2$, the existence of a graph H' on at most $\mathcal{O}(\bar{d}^{\delta+2})$ vertices with minimum degree at least $\bar{d}$, maximum degree at most $2\bar{d}$ and girth $g \geq \delta + 2$ follows from Erdős' bound on the size of cages [18] (see [19] for an English translation).

Given such a graph H', we construct H as follows. We replace each vertex v in H' by an in-vertex v_{in} and an out-vertex v_{out}. For every edge $\{u, v\}$ of H' we insert the edges $\{u_{\text{in}}, v_{\text{out}}\}$ and $\{v_{\text{in}}, u_{\text{out}}\}$. Now, H has a girth which is at least as large as the girth of H', since any cycle in the new graph projects to a closed walk with no immediate backtracking and without trivial steps, which contains a cycle in the original graph. Note that H is bipartite and that the degree of both the in- and out-vertices is equal to the degree of the vertex they replace. Note that, since every vertex is replaced by two other vertices, the final number of vertices is in $\mathcal{O}(\bar{d}^{\delta+2})$. If H is not connected, we replace it by one of its connected components. □

Let H be a connected bipartite n'-vertex graph with minimum degree at least $\bar{d}$, maximum degree at most $2\bar{d}$, and girth at least $\delta + 2$ as given by Lemma 1. Suppose the partition classes have size n_1 and n_2. We fix orders $(u_1, \ldots, u_{n_1})$ and $(v_1, \ldots, v_{n_2})$ for the vertices in each of the bipartition classes. As in the proof of Lemma 1 we call the vertices in the bipartition classes in-vertices and out-vertices respectively.

We order the edges by the lexicographical ordering that satisfies $\{u, v\} < \{u', v'\}$ if $v < v'$ or $(v = v'$ and $u < u')$.

In H we now replace each in-vertex and each out-vertex by a release gadget and collection gadget, respectively. Our final construction will have edges with two types of weights, namely 1 and $w > \max\{1, (2\bar{d}+1)/(\delta+1)\}$.

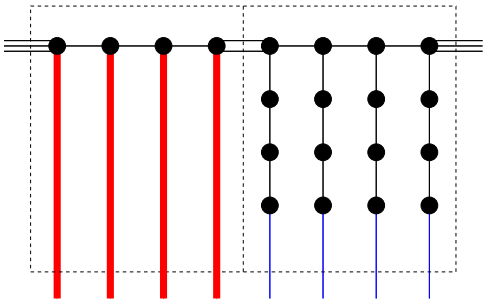

Fig. 2. The release gadget for a vertex of degree $d = 4$. The left part and the right part are connected by a center path of length $(\delta + 1)w - d$ consisting of unit-weight edges. This path is depicted as a double edge. The gadget is also connected to the left and right to other release gadgets by blocking path of length $(\delta + 1)w$ with unit weights. These paths are depicted as triple edges.

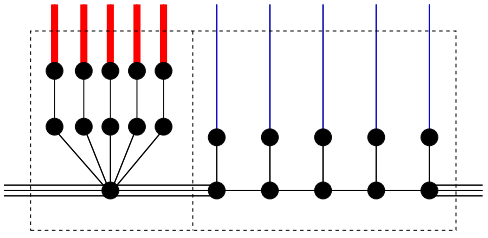

Fig. 3. The collection gadget for a vertex of degree $d = 5$. The left part has a vertex of degree d and is the end of d paths of length two. The right part consists of d vertices that lie on a path. Each vertex on the path has an additional neighbor. The two parts are connected by a blocking path in the center. A blocking path joins each part with another collection gadget.

Description of the release gadgets. Fig. 2 depicts a release gadget for a vertex of degree 4. In general a release gadget consists of two parts, which we call the left part and right part. A gadget replacing a vertex of degree d consists in the left part of d vertices forming a path. Each of these vertices is attached to a heavy (red) edge of weight w that has an endpoint in some collection gadget. The right part contains d vertices forming a path. Each of these vertices is incident with an attached *release path* of length $d - 1$. The endpoints of these paths are incident with a (blue) edge that ends in a collection gadget. The two parts are joined by a *center path* of length $\lfloor(\delta + 1)w - d\rfloor$ (depicted by a double edge) with unit-weight edges. Finally, each part has a *blocking path* (depicted as triple edges) of length $\lfloor(\delta + 1)w\rfloor$ with unit-weight edges, by which it is connected to other release gadgets.

The crucial property of a release gadget is the following. The length of the center path is chosen such that the i-th heavy edge may be blocked by the first edge of the i-th release path, but not by the $(i + 1)$-th release path (both times counting from the left to right). Once the exploration of a release path has begun, the algorithm will finish the exploration of the entire release path before exploring any other edges.

Thus, the i-th heavy edge of the gadget is blocked if one of the release paths $1, \ldots, i$ has not yet been explored. If a release path has been completely explored, we also say that the release has been triggered. Suppose in some release gadget all releases $1, \ldots, i$ have been triggered, but the i-th heavy edge is still blocked. This situation implies that there is a path to some unit-weight boundary edge which exits the gadget via another heavy edge. Indeed, the blocking paths are sufficiently long to prevent other release gadgets from interfering with this fact. We will show later that at the moment release i is triggered such paths exiting via heavy edges do not exist.

It will be clear later that when a release gadget is entered for the first time, this happens via the blocking path to the right of the gadget. Assuming this for now, we can require that the online algorithm traverses the gadget from right to left, without entering the release paths. Indeed, whenever there is a choice among edges of equal weight, we can adversarially choose the edge that is traversed next.

Description of the collection gadgets. Fig. 3 depicts a collection gadget for a vertex of degree 5. In general a collection gadget consists of a left and a right part. For a gadget replacing a vertex of degree d, the left part has one vertex of degree d incident with d paths of length 2. The ends of these paths are incident with heavy (red) edges emanating from release gadgets. The right part of a collection gadget contains d vertices inducing a path. Each vertex on the path is adjacent to another vertex which itself is incident with a (blue) edge emanating from a right part of a release gadget.

Three blocking paths (triple edges) of length $\lfloor(\delta + 1)w\rfloor$ join the parts with each other and with other collection gadgets.

We will see that when a collection gadget is first entered, this happens via the blocking path to the left. We can then require adversarially that the online algorithm traverses from the left part directly to the right without exploring the left part. Then, on entering a vertex in the right, it deviates from the main path to explore the respective blue edge. We will argue that the algorithm will then return via a corresponding heavy edge. It then backtracks and subsequently explores the next vertex of the right part, and so on. Before leaving the gadget to the right, the gadget has been completely explored.

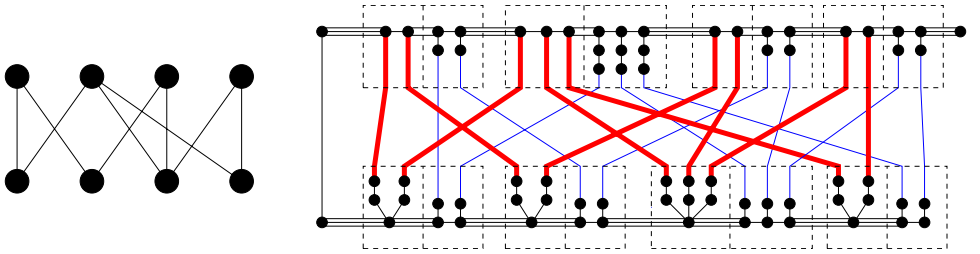

Fig. 4. The assembly of the gadgets. A base graph H (left), and the resulting graph with linked replacement gadgets (right) showing release gadgets on top and collection gadget at the bottom.

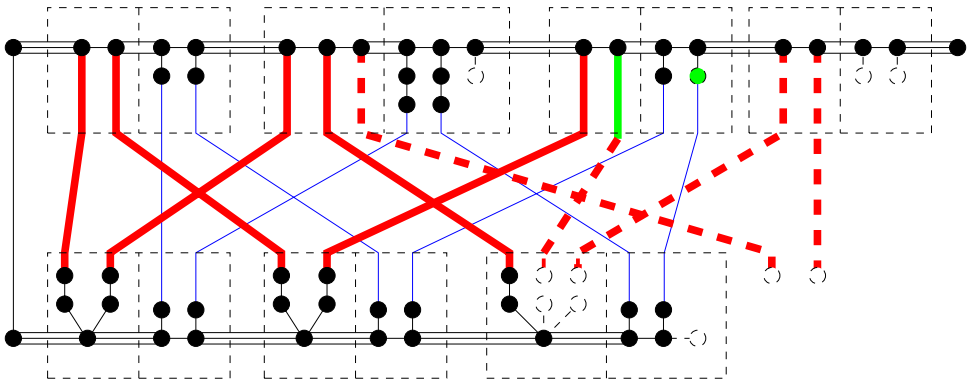

Fig. 5. A snap-shot during an execution of Blocking executed on the graph from Fig. 4. Unexplored boundary vertices and boundary edges are shown dashed. Two collection gadgets have been completely explored, the green vertex (with dashed boundary) has been explored last. It releases the second edge of its release gadget (shown partially green, partially dashed).

Assembly of the gadgets according to the base graph H. To assemble the gadgets using the base graph H (see Fig. 4), we join the release gadgets according to the order of in-vertices along the blocking paths (triple lines). The same is done with the collection gadgets, with respect to the order of out-vertices. The right blocking path of the last (rightmost) release gadget is connected to a single vertex, the starting vertex, that we add to the graph. The left blocking path of the first (leftmost) release gadget and the first (leftmost) collection gadget are joined by two added, adjacent vertices.

The (red and blue) edges that run between the gadgets correspond to the edges in H. Heavy (red) edges of weight w run from a left part of a release gadget to a left part of a collection gadget. Blue edges of weight 1 run from a right part of a release gadget to a right part of a collection gadget.

In the lexicographical order of the edges defined above, we insert for each edge of H a heavy (red) edge and a blue edge. To insert a heavy edge corresponding to the edge (u, v) of H, we connect the leftmost unused vertex in the left part of the release gadget corresponding to u with the leftmost unused vertex in the left part of the collection gadget of v. To insert the blue edge, we connect the leftmost unused vertex in the right part of the release gadget corresponding to u with the leftmost unused vertex of the right part of the collection gadget of v.

Inserting the heavy edges in this ordering has the consequence that the ordering of the edges is exactly the ordering of their end vertices in the collection gadgets from left to right. Furthermore, within each release gadget, the heavy edges from left to right are also in the lexicographic order.

The tour traversed by the algorithm. Beginning at the starting vertex, we may require adversarially that the algorithm first traverses all release gadgets without exploring any release path. Then, via the two additional vertices on the left, the leftmost collection gadget is entered from the left, and the exploration continues into its right part. Subsequently release paths are triggered, one at a time. In the following we prove that the algorithm traverses all heavy edges of H. The lexicographic order defined on the edges is the order in which these edges are traversed. All of them are traversed from a release gadget to a collection gadget. The blue edges, each used to trigger a traversal of a heavy edge, are traversed from a collection gadget to a release gadget. Recall that due to the length of the center path connecting the right and left parts of a release gadget, a heavy edge is blocked, unless its corresponding release has been triggered.

Lemma 2. *The heavy (red) edges are traversed in the lexicographic order of the edges of the base graph. Whenever a release is triggered, the corresponding heavy edge e_r becomes unblocked and is explored subsequently.*

Fig. 5 shows a typical situation during the execution of the algorithm, when an edge becomes unblocked.

Proof of Lemma 2. Inductively we assume that all release paths that correspond to edges that appear earlier than e_r in the ordering of edges have been completely explored, and all release paths that appear later than e_r are completely unexplored.

Algorithm 2 Exploration algorithm hDFS(G, u, w)

Input: A partially explored graph G, a vertex u of G that is visited for the first time, and a weight $w \in \mathbb{R}_{\geq 0} \cup \{\infty\}$.

while there is a weight $w' < w$ such that w' occurs in comp($G_{\leq w'}$, u) but comp($G_{\leq w'}$, u) is not completely explored **do**
  hDFS(G, u, w')
end while
choose a minimal spanning tree of comp($G_{<w}$, u) and order all vertices according to a depth first search in this spanning tree
while there is a boundary edge (u', y') of weight w with $u' \in$ comp($G_{<w}$, u) **do**
  let (u', y') be a boundary edge of weight w with $u' \in$ comp($G_{<w}$, u) such that u' is minimal with respect to the ordering
  traverse a shortest path to y'
  hDFS(G, y', w)
end while
traverse a shortest path to u

A heavy weight edge can only be blocked by an edge of weight 1. Thus, for a heavy edge to be blocked, there has to be a path of length at most $(\delta + 1)w - 1$ to a boundary edge of weight 1. To show the claim, we show that no such path exists for edge e_r.

To do so, we analyze where a hypothetical boundary edge of such a path may be situated in the graph. Observe that the length of blocking paths (triple edge) is chosen such that they cannot be traversed to reach a boundary edge within a distance of $(\delta + 1)w - 1$. Thus, only two possibilities have to be ruled out:

1. There is a path to a boundary edge that can be reached by a path of length $(1 + \delta)w - 1$, which traverses a center path (double edge).
2. There is a path to a boundary edge that uses heavy edges, but otherwise is completely contained in left parts of gadgets.

To rule out Possibility 1, observe that any path that uses a double line to cross from a left part of a release gadget to a right part, and then uses a complete release path is longer than $(\delta + 1)w - 1$. Moreover, since release paths are either completely explored or the corresponding heavy edge has by induction not been triggered, for every unexplored edge in the right part of a release gadget, all explored heavy edges in the left part are further away than $(\delta + 1)w - 1$.

To rule out Possibility 2, note that the only boundary edges of weight 1 situated in the left part of a gadget are contained in the currently used collection gadget. All other left parts of gadgets have been completely explored or not explored at all. Thus, any path staying in the left parts of the gadgets that leads to a boundary edge in the left part of the currently used collection gadget will, together with e_r, project to a cycle in the graph H. Since the girth of H is at least $\delta + 2$, the path has to use at least $\delta + 1$ heavy edges and is thus of length more than $(\delta + 1) \cdot w$.

We have shown that the heavy edge e_r becomes unblocked when its release is triggered. The algorithm thus explores e_r, returns to the release path corresponding to e_r, backtracks, and continues to trigger the release corresponding to the next heavy edge. □

The previous lemma implies in particular that Algorithm Blocking traverses all heavy edges. This allows us to show that the online algorithm does not have a constant competitive ratio.

Theorem 3. *The competitive ratio of* $\mathsf{Blocking}_\delta$ *for* $\delta \in \mathbb{R}^+$ *is in* $\Omega(n^{1/(\delta+4)})$.

Proof. Consider a graph that is obtained from the replacement construction from a base graph H on n' vertices with minimum degree $\bar{d}$, maximum degree at most $2\bar{d}$, and girth at least $\delta + 2$ (Lemma 1). Including blocking paths, the number of unit-weight edges in a release gadget corresponding to a vertex v of degree $d(v)$ is $\mathcal{O}(d(v)^2) + \mathcal{O}(\delta w) \subseteq \mathcal{O}(\bar{d}^2) + \mathcal{O}(\delta w)$. This bound also holds for collection gadgets. Thus, for fixed δ, the resulting graph has a minimum spanning tree of size $\mathcal{O}(n'\bar{d}^2) + \mathcal{O}(n'w)$. Since $\mathsf{Blocking}_\delta$ traverses all heavy edges (Lemma 2), it incurs a cost of $\Omega(\bar{d}n'w)$. Thus, its competitive ratio is in $\Omega(\bar{d}w/(\bar{d}^2 + w))$. For fixed $\bar{d}$ we can choose $w \in \Theta(\bar{d}^2)$ so that this ratio is $\Omega(\bar{d})$. Note that by fixing $\bar{d}$ first, we can additionally achieve that $w > (2\bar{d} + 1)/(\delta + 1)$, which is required for the center path to be of positive length and thus necessary for the construction to be realizable. By Lemma 1 we can assume that $n' \in \mathcal{O}(\bar{d}^{\delta+2})$. Thus the final graph has $n \in \mathcal{O}(\bar{d}^{\delta+2}\bar{d}^2)$ vertices. Therefore, the competitive ratio is in $\Omega(\bar{d}) \subseteq \Omega(n^{1/(\delta+4)})$. □

5. Graphs with a bounded number of distinct weights

We describe an algorithm that has constant competitive ratio, when the input graphs have a bounded number of distinct weights. To do so, we first define components of graphs induced by edges of restricted weight.

Definition 3. For any graph G, weight w, and vertex u, let comp($G_{\leq w}$, u) be the connected component of the subgraph of G comprised of all edges of weight at most w containing u. The subgraph comp($G_{<w}$, u) is defined similarly using edges of weight strictly less than w.

Our algorithm *hierarchical depth first search* (hDFS), formally given by Algorithm 2, explores the graph $\text{comp}(G_{\leq w}, u)$ for any weight w. The weight $w \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ is provided as a parameter. The algorithm is based on a depth first search in the graph $\text{comp}(G_{\leq w}, u)$. However, whenever a new vertex of this component is encountered, it first explores $\text{comp}(G_{<w}, u)$. The algorithm then intuitively simulates the depth first search in the graph $G/\text{comp}(G_{<w}, u)$. Here G/H denotes the graph obtained from G by contracting the subgraph H of G to a single point. To ensure that the total length traversed within $H = \text{comp}(G_{<w}, u)$ is not too large, the boundary edges leaving H are explored according to a specific order. This order is obtained by computing a depth first search on a minimum spanning tree of $\text{comp}(G_{<w}, u)$.

The computation of $\text{comp}(G_{<w}, u)$ can be reduced to recursive calls of the algorithm itself with parameters smaller than w. This is possible due to the following basic observation:

Lemma 3. *The component* $\text{comp}(G_{<w}, u)$ *is completely explored if and only if there is no boundary edge of weight smaller than* w *with an end-vertex in component* $\text{comp}(G_{<w}, u)$.

To explore the entire graph starting in vertex s, we call Algorithm 2 as $\text{hDFS}(G_s, s, \infty)$, where G_s is the partially explored graph in which only s has been visited so far.

Theorem 4. *Algorithm* hDFS *is* $2k$*-competitive on graphs with at most* k *distinct weights.*

Proof. We first show that all vertices are explored. To prove this it suffices to show that the call $\text{hDFS}(G, s, w)$ explores $\text{comp}(G_{\leq w}, s)$. Suppose that there remain unexplored edges with weight w or less in component $\text{comp}(G_{\leq w}, s)$ after call $\text{hDFS}(G, s, w)$. At least one of these edges is a boundary edge, say (u, v), with v unexplored and with weight $w' \leq w$. Consider the last call of Algorithm 2 for some node $z \in \text{comp}(G_{\leq w}, s)$ with parameter $w'' \geq w'$ before u is explored, i.e., we consider the call $\text{hDFS}(G, z, w'')$. After node u is explored, the edge (u, v) with weight $w' \leq w''$ is known to be a boundary edge in $\text{comp}(G_{\leq w''}, z)$. Thus, continuing the execution of $\text{hDFS}(G, z, w'')$ eventually invokes the call $\text{hDFS}(G, u, w')$. This call causes v to be explored and gives a contradiction.

Let *MST* be a minimum spanning tree of G. To show that the algorithm is $2k$-competitive, we show that for each $w < \infty$ the sum of all traversals made in calls with parameter w is at most $2|MST|$. For this it suffices to show: if F is a sub-forest of G that contains edges of weight at most w such that for each vertex u the graph $\text{comp}(F_{<w}, u)$ is a minimum spanning tree of $\text{comp}(G_{<w}, u)$, then F is contained in a minimum spanning tree of G. Finally note that the outer call with parameter $w = \infty$ does not incur any costs. □

The proof shows that the set of all edges traversed by hDFS forms a minimum spanning tree of G.

For graphs with arbitrary weights, we adapt the algorithm by rounding each edge weight to the nearest power of 2 and simulating the exploration on this altered graph.

Theorem 5. *Algorithm* hDFS *with weights rounded to powers of 2 has competitive ratio* $\Theta(\log(n))$.

Proof. Let G' be the graph obtained from an input graph G by rounding the edge weights up to the nearest power of 2. Consider a minimum spanning tree MST' in G'. For $i = 1, \ldots, k$, let w_i denote the i-th smallest edge weight in MST', and let t_i denote the number of edges with weight w_i in MST'. By definition, $\sum_{i=1}^{k} t_i = n - 1$, and $|MST'| = \sum_{i=1}^{k} t_i \cdot w_i$. Note that $|MST'|$ is within a factor of two of the size of a minimum spanning tree of the original graph G, our standard lower bound on the optimal tour.

The online algorithm traverses the edges of MST'. The crucial observation is that an edge of i-th largest weight, w_{k-i+1}, is traversed at most $2i$ times. Thus, the total cost of the algorithm is at most $\sum_{i=1}^{k} 2(k-i+1)t_i \cdot w_i$. We express this cost as a function of $|MST'|$ and show that it is in $\mathcal{O}(\log(n) \cdot |MST'|)$. In the case that $k \leq \log(n)$, this is trivially true. For the case that $k > \log(n)$, we bound the cost for small and large edges separately:

$$\sum_{i=1}^{k} 2(k-i+1)t_i \cdot w_i = \sum_{i=1}^{k-\lceil \log(n) \rceil} 2(k-i+1)t_i \cdot w_i + \sum_{i=k-\lceil \log(n) \rceil + 1}^{k} 2(k-i+1)t_i \cdot w_i.$$

As a function of integers, $(k-i+1)w_i$ is monotone increasing for $i \leq k$. Thus, we can bound the first summand by $2n(\lceil \log(n) \rceil + 1)w_{k-\lceil \log(n) \rceil}$. Since the ratio between any two different weights w_i and w_j is at least 2^{i-j}, we have $w_{k-\lceil \log(n) \rceil} \leq w_k/n$. Hence, the first summand is at most $2(\log(n) + 2)w_k \leq 2(\log(n) + 2) \cdot |MST'|$. The second summand is bounded by $2\lceil \log(n) \rceil \cdot |MST'|$ because $\sum_{i=k-\lceil \log(n) \rceil + 1}^{k} t_i \cdot w_i \leq |MST'|$, and thus the upper bound follows.

It remains to show that this upper bound is tight. Let $L = 2^{k+1} - 2k$, for some integer $k > 1$. Our lower bound graph is a path with $n = L + 2k = 2^{k+1}$ vertices and k distinct weights. It is constructed as follows. The start vertex is the left endpoint of a path of unit-weight edges of total length $L - 1$. At both ends of this path we repeatedly attach edges of geometrically increasing weights $1, 2, 4, \ldots, 2^{k-1}$. An optimal tour is obtained by exploring first the chain to the left of the start vertex, then the one to the right, and returning to the origin. This tour traverses each edge exactly twice, and the tour thus has total cost $2(L-1) + 4\sum_{i=0}^{k-1} 2^i \leq 2^{k+3}$.

For every $i > 1$ our algorithm explores both edges of weight 2^i before exploring any edge of larger weight. In particular the algorithm traverses the path of unit-weight edges of length L at least $k - 1$ times. Thus, the total cost for exploring the entire graph is at least $(k-1)(2^{k+1} - 2k) \geq (k-1)2^k$.

This gives a worst case ratio linear in k, which implies the desired bound by our choice of $n = 2^{k+1}$. □

6. Concluding remarks

Our main result is a non-trivial graph construction which proves that Algorithm Blocking does not have a constant competitive ratio on arbitrary graphs. This answers a longstanding open question. Nevertheless, the result does not generally rule out online algorithms with constant competitive ratio. In particular, our construction involves only two distinct types of weights, and thus our new Algorithm hDFS has constant competitive ratio. However, at present, there is no candidate for an algorithm that may achieve a constant competitive ratio on general graphs. Of course showing that no such algorithm exists might require a construction even more complicated than the one presented in this paper. For such a result it might be helpful to use the fact that one can equivalently consider the exploration model in which the label of a vertex is only revealed upon arrival at the vertex. This can be seen by replacing each vertex by a star with edges of small weight, and linking the previous neighbors to the outer vertices of the star.

Acknowledgment

This work was supported by the National Research Fund, Luxembourg and cofunded under the Marie Curie Actions of the European Commission (FP7-COFUND).

References

[1] S. Albers, M.R. Henzinger, Exploring unknown environments, SIAM Journal on Computing 29 (4) (2000) 1164–1188.
[2] Y. Asahiro, E. Miyano, S. Miyazaki, T. Yoshimuta, Weighted nearest neighbor algorithms for the graph exploration problem on cycles, Information Processing Letters 110 (3) (2010) 93–98.
[3] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, M. Talamo, Algorithms for the on-line travelling salesman, Algorithmica 29 (4) (2001) 560–581.
[4] B. Awerbuch, M. Betke, R.L. Rivest, M. Singh, Piecemeal graph exploration by a mobile robot, Information and Computation 152 (2) (1999) 155–172.
[5] R.A. Baeza-Yates, J.C. Culberson, G.J.E. Rawlins, Searching in the plane, Information and Computation 106 (2) (1993) 234–252.
[6] M.A. Bender, D.K. Slonim, The power of team exploration: Two robots can learn unlabeled directed graphs, in: Proceedings of the 35th Annual Symposium on Foundations of Computer Science, FOCS'94, IEEE, 1994.
[7] P. Berman, On-line searching and navigation, in: Online Algorithms: The State of the Art, in: Lecture Notes in Computer Science, vol. 1442, Springer, 1998.
[8] M. Betke, R.L. Rivest, M. Singh, Piecemeal learning of an unknown environment, Machine Learning 18 (1995) 231–254.
[9] V. Bonifaci, Models and algorithms for online server routing, Ph.D. Thesis, Sapienza Università di Roma, Italy, 2007.
[10] A. Borodin, R. El-Yaniv, Online Computation and Competitive Analysis, Cambridge University Press, 1998.
[11] X. Deng, C. H. Papadimitriou, Exploring an unknown graph (extended abstract), in: Proceedings of the 31st Annual Symposium on Foundations of Computer Science, FOCS'90, IEEE, 1990.
[12] A. Dessmark, A. Pelc, Optimal graph exploration without good maps, Theoretical Computer Science 326 (1–3) (2004) 343–362.
[13] C.A. Duncan, S.G. Kobourov, V.S.A. Kumar, Optimal constrained graph exploration, ACM Transactions on Algorithms 2 (2006) 380–402.
[14] M. Dynia, J. Kutylowski, F. der Heide, C. Schindelhauer, Smart robot teams exploring sparse trees, in: R. Krlovic, P. Urzyczyn (Eds.), Mathematical Foundations of Computer Science, MFCS 2006, in: Lecture Notes in Computer Science, vol. 4162, Springer, Berlin/Heidelberg, 2006, pp. 327–338.
[15] M. Dynia, J. Lopuszanski, C. Schindelhauer, Why robots need maps, in: G. Prencipe, S. Zaks (Eds.), Proceedings of the 14th International Colloquium on Structural Information and Communication Complexity, SIROCCO'07, in: Lecture Notes in Computer Science, vol. 4474, Springer, 2007.
[16] H. A. Eiselt, M. Gendreau, G. Laporte, Arc routing problems, part I: the chinese postman problem, Operations Research 43 (2) (1995) 231–242.
[17] P. Erdős, Graph theory and probability, Canadian Journal of Mathematics 11 (1959) 34–38.
[18] P. Erdős, H. Sachs, Reguläre Graphen gegebener Taillenweite mit minimaler Knotenzahl, Zeitschrift der Martin-Luther-Universität Halle-Wittenberg, Mathematisch-Naturwissenschaftliche Reihe 12 (1963) 251–257.
[19] G. Exoo, R. Jajcay, Dynamic cage survey, Electronic Journal of Combinatorics (2011) DS16.
[20] R. Fleischer, G. Trippen, Experimental studies of graph traversal algorithms, in: K. Jansen, M. Margraf, M. Mastrolilli, J.D.P. Rolim (Eds.), Proceedings of the Second International Workshop on Experimental and Efficient Algorithms, WEA'03, in: Lecture Notes in Computer Science, vol. 2647, Springer, 2003.
[21] R. Fleischer, G. Trippen, Exploring an unknown graph efficiently, in: Brodal G.S., Leonardi S. (Eds.), Proceedings of the 13th Annual European Symposium on Algorithms, ESA'05, in: Lecture Notes in Computer Science, vol. 3669, Springer, 2005.
[22] P. Fraigniaud, L. Gąsieniec, D.R. Kowalski, A. Pelc, Collective tree exploration, Network 48 (2006) 166–177.
[23] P. Fraigniaud, D. Ilcinkas, A. Pelc, Impact of memory size on graph exploration capability, Discrete Applied Mathematics 156 (12) (2008) 2310–2319.
[24] S. Gal, Search Games, Academic Press, 1980.
[25] L. Gasieniec, T. Radzik, Memory efficient anonymous graph exploration, in: H. Broersma, T. Erlebach, T. Friedetzky, D. Paulusma (Eds.), Proceedings of the 34th International Workshop on Graph-Theoretic Concepts in Computer Science, WG'08, in: Lecture Notes in Computer Science, vol. 5344, 2008.
[26] S.K. Ghosh, R. Klein, Online algorithms for searching and exploration in the plane, Computer Science Review 4 (4) (2010) 189–201.
[27] G. Gutin, A.P. Punnen, The Traveling Salesman Problem and Its Variations, Springer, 2002.
[28] C.A.J. Hurkens, G.J. Woeginger, On the nearest neighbor rule for the traveling salesman problem, Operations Research Letters 32 (1) (2004) 1–4.
[29] P. Jaillet, M.R. Wagner, Online vehicle routing problems: a survey, in: R. Sharda, B. Golden, S. Raghavan, E. Wasil (Eds.), The Vehicle Routing Problem: Latest Advances and New Challenges, in: Operations Research/Computer Science Interfaces, vol. 43, Springer US, 2008, pp. 221–237.
[30] B. Kalyanasundaram, K. Pruhs, Constructing competitive tours from local information, Theoretical Computer Science 130 (1) (1994) 125–138.
[31] T. Kamphans, Models and algorithms for online exploration and search, Ph.D. Thesis, University of Bonn, 2005.
[32] S. Kutten, Stepwise construction of an efficient distributed traversing algorithm for general strongly connected directed networks or: traversing one way streets with no map, in: ICCC, 1988.
[33] S. Kwek, On a simple depth-first search strategy for exploring unknown graphs, in: F.K.H.A. Dehne, A. Rau-Chaplin, J.-R. Sack, R. Tamassia (Eds.), Proceedings of the 5th International Workshop on Algorithms and Data Structures, WADS'97, in: Lecture Notes in Computer Science, vol. 1272, Springer, 1997.
[34] E.L. Lawler, J.K. Lenstra, A.H.G.R. Kan, D.B. Shmoys, The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, John Wiley & Sons, New York, 1985.
[35] S. Miyazaki, N. Morimoto, Y. Okabe, The online graph exploration problem on restricted graphs, IEICE Transactions on Information and Systems E92.D (9) (2009) 1620–1627.
[36] B. Mohar, C. Thomassen, Graphs on surfaces, in: Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, 2001.
[37] P. Panaite, A. Pelc, Exploring unknown undirected graphs, Journal of Algorithms 33 (2) (1999) 281–295.
[38] C. Papadimitriou, M. Yannakakis, Shortest paths without a map, Theoretical Computer Science 84 (1) (1991) 127–150.
[39] N. Rao, S. Kareti, W. Shi, S. Iyengar, Robot navigation in unknown terrains: introductory survey of nonheuristic algorithms, Tech. Rep. ORNL/TM-12410, Oak Ridge National Laboratory, 1993.
[40] D.J. Rosenkrantz, R.E. Stearns, P.M. Lewis II, An analysis of several heuristics for the traveling salesman problem, SIAM Journal on Computing 6 (3) (1977) 563–581.