Tight Bounds for Undirected Graph Exploration with Pebbles and Multiple Agents

YANN DISSER, TU Darmstadt, Germany
JAN HACKFELD and MAX KLIMM, HU Berlin, Germany

We study the problem of deterministically exploring an undirected and initially unknown graph with n vertices either by a single agent equipped with a set of pebbles or by a set of collaborating agents. The vertices of the graph are unlabeled and cannot be distinguished by the agents, but the edges incident to a vertex have locally distinct labels. The graph is explored when all vertices have been visited by at least one agent. In this setting, it is known that for a single agent without pebbles $\Theta(\log n)$ bits of memory are necessary and sufficient to explore any graph with at most n vertices. We are interested in how the memory requirement decreases as the agent may mark vertices by dropping and retrieving distinguishable pebbles or when multiple agents jointly explore the graph. We give tight results for both questions showing that for a single agent with constant memory $\Theta(\log \log n)$ pebbles are necessary and sufficient for exploration. We further prove that using collaborating agents instead of pebbles does not help as $\Theta(\log \log n)$ agents with constant memory each are necessary and sufficient for exploration.

For the upper bounds, we devise an algorithm for a single agent with constant memory that explores any n-vertex graph using $O(\log \log n)$ pebbles, even when n is not known *a priori*. The algorithm terminates after polynomial time and returns to the starting vertex. We further show that the algorithm can be realized with additional constant-memory agents rather than pebbles, implying that $O(\log \log n)$ agents with constant memory can explore any n-vertex graph. For the lower bound, we show that the number of agents needed for exploring any graph with at most n vertices is already $\Omega(\log \log n)$ when we allow each agent to have at most $O((\log n)^{1-\varepsilon})$ bits of memory for any $\varepsilon > 0$. Our argument also implies that a single agent with sublogarithmic memory needs $\Theta(\log \log n)$ pebbles to explore any n-vertex graph.

CCS Concepts: • **Mathematics of computing** → **Graph algorithms**; • **Theory of computation** → *Graph algorithms analysis*; • **Computing methodologies** → Distributed algorithms;

Additional Key Words and Phrases: Graph exploration, pebbles, space efficiency, multi-agent system

ACM Reference format:
Yann Disser, Jan Hackfeld, and Max Klimm. 2019. Tight Bounds for Undirected Graph Exploration with Pebbles and Multiple Agents. *J. ACM* 66, 6, Article 40 (October 2019), 41 pages.
https://doi.org/10.1145/3356883

Results concerning exploration with pebbles appeared in preliminary form in Reference [17].

Authors' addresses: Y. Disser, TU Darmstadt, Graduate School CE, Dolivostraße 15, 64293, Darmstadt, Germany; email: disser@mathematik.tu-darmstadt.de; J. Hackfeld and M. Klimm, HU Berlin, School of Business and Economics, Spandauer Straße 1, 10178, Berlin, Germany; emails: {jan.hackfeld, max.klimm}@hu-berlin.de.

0004-5411/2019/10-ART40 $15.00
https://doi.org/10.1145/3356883

1 INTRODUCTION

The exploration of unknown graphs subject to space or time bounds is a fundamental problem with applications to robot navigation, Internet crawling, and image recognition. Its pivotal role within the theory of computation stems from the fact that it is a natural abstraction of a process of computing where nodes correspond to states, edges correspond to possible state transitions, and the goal is to find an accepting state when starting in a given initial state. Single agent exploration then corresponds to computing with a single processing unit while exploration with multiple agents corresponds to parallel computing. In this context, graph exploration and traversal problems have proven to be useful to study the relationship between probabilistic and deterministic space-bounded algorithms [36].

The time and space complexity of undirected graph exploration by a *single* agent is fairly well understood. Aleliunas et al. [2] showed that a random walk of length $n^5 \log n$ visits all vertices of any n-vertex graph with high probability. When n is known, it is thus possible to use a counter that keeps track of the number of steps taken to obtain a probabilistic algorithm that explores any graph of size n in polynomial time and $O(\log n)$ space with high probability. In a breakthrough result, Reingold [33] showed that the same time and space complexity can be achieved by a *deterministic* algorithm. Both the randomized algorithm of Aleliunas et al. and the deterministic algorithm of Reingold work for anonymous graphs where vertices are indistinguishable. Logarithmic memory is in fact necessary to explore all anonymous graphs with n vertices; see Fraigniaud et al. [22].

Already the early literature on graph exploration problems is rich with examples where exploration is made feasible or the time or space complexity of exploration by a single agent can be decreased substantially by either allowing the agent to mark vertices with pebbles or by cooperating with other agents. For instance, two-dimensional mazes can be explored by a single agent with finite memory using two pebbles [7, 8, 37] or by two cooperating agents with finite memory [7], while a single agent with finite memory (and even a single agent with finite memory and a single pebble) does not suffice [9, 25]. Directed anonymous graphs can be explored in polynomial time by two cooperating agents [5] or by a single agent with $\Theta(\log \log n)$ indistinguishable pebbles and $O(n^2 \Delta \log n)$ bits of memory [4, 21], where Δ is the maximum out-degree in the graph. Note that a single agent needs at least $\Omega(n \log \Delta)$ bits of memory in this setting, even if it is equipped with a linear number of indistinguishable pebbles [21], and it needs exponential time for exploration if it only has a constant number of pebbles and no upper bound on the number of vertices is known [4].

Less is known regarding the complexity of general undirected graph exploration by more than one agent or an agent equipped with pebbles. Rollik [34] showed that there are finite graphs, henceforth called *traps*, that a finite set of k agents each with a finite number s of states cannot explore. Fraigniaud et al. [24] revisited Rollik's construction and observed that the traps have $\tilde{O}(s \uparrow\uparrow (2k+1))$ vertices, where $a \uparrow\uparrow b := a^{a^{\cdot^{\cdot^{a}}}}$ with b levels in the exponent and $\tilde{O}$ suppresses lower order terms. Fraigniaud et al. also gave an improved upper bound of $\tilde{O}(s \uparrow\uparrow (k+1))$. Cook and Rackoff [13] showed that even in a stronger model with a constant number of agents that have full knowledge of their states even when they do not share a location, and can in one step jump to the location of any other agent, there are graphs that cannot be explored.

While it is plausible that an agent with s states and p pebbles is less powerful than a set of $p+1$ agents with s states each, no better bounds for a single agent with pebbles were known. Even more striking is the lack of any non-trivial upper bounds for the exploration with several agents or the single agent exploration with pebbles for undirected graphs. Specifically, there was no algorithm known that explores an undirected graph with sublogarithmic space when more than one agent and/or pebbles are allowed.

1.1 Our Results

We give tight bounds for both the space complexity of undirected graph exploration by a single agent with pebbles, as well as by a set of cooperating agents.

1.1.1 Results for Exploration with Pebbles. For the exploration of a graph by a single agent with constant memory, we show that $\Theta(\log\log n)$ distinguishable pebbles are necessary and sufficient to explore all undirected anonymous graphs with at most n vertices.

Our proof of the upper bound is constructive, i.e., we devise an algorithm that explores any graph with n vertices using $O(\log\log n)$ pebbles (Corollary 3.7). Our algorithm terminates after having explored the graph and returns to the starting vertex with all pebbles. We further show that the exploration time, i.e., the number of edge traversals of the agent, is polynomial in the size of the graph. Our algorithm does not require n to be known and gradually increases the number of used pebbles during the course of the algorithm such that for any n-vertex graph at most $f(n)$ pebbles are used where $f(n) \in O(\log\log n)$.

For a lower bound, we show that a single agent with sublogarithmic memory (more precisely $O((\log n)^{1-\epsilon})$ bits of memory for an arbitrary constant $\epsilon > 0$) already needs $\Omega(\log\log n)$ pebbles for exploring every graph with n vertices (Corollary 4.15). Our results fully characterize the tradeoff between the memory and the number of pebbles of an agent needed for exploration. It turns out that this tradeoff is governed by two thresholds. When the agent has $\Omega(\log n)$ bits of memory, no pebbles are needed at all. But as soon as the memory is $O((\log n)^{1-\epsilon})$ already $\Omega(\log\log n)$ pebbles are needed to explore all n-vertex graphs. However, with $\Omega(\log\log n)$ pebbles already a constant number of bits of memory are sufficient for exploration.

1.1.2 Results for Multi-agent Exploration. For collaborative graph exploration, we show that $\Theta(\log\log n)$ agents with constant memory are necessary and sufficient to explore all undirected anonymous graphs with at most n vertices.

To this end, we first show that a set of $2p+2$ agents, with a constant set of states each, can reproduce the exploration by a single agent with a constant number of states and p pebbles (Lemma 2.3). This result allows us to rephrase our single agent exploration algorithm with $O(\log\log n)$ pebbles as a multi-agent exploration algorithm with $O(\log\log n)$ agents and constant memory each (Corollary 3.8). As a perhaps surprising result, we show that this is optimal in terms of the asymptotic number of agents. To prove this lower bound, we construct a family of graphs with $O(s^{2^{5k}})$ vertices that trap any set of k agents with s states each (Theorem 4.13). Our construction exhibits dramatically smaller traps with only a doubly exponential number of vertices compared to the traps of size $\tilde{O}(s \uparrow\uparrow (2k+1))$ and $\tilde{O}(s \uparrow\uparrow (k+1))$ due to Rollik [34] and Fraigniaud et al. [24], respectively. As a consequence of our improved bound on the size of the trap, we are able to show that, even if we allow $O((\log n)^{1-\epsilon})$ bits of memory for an arbitrary constant $\epsilon > 0$ for every agent, the number of agents needed for exploration is at least $\Omega(\log\log n)$ (Theorem 4.14). This construction also yields the lower bound for a single agent with pebbles, as $p+1$ agents with $O((\log n)^{1-\epsilon})$ bits of memory each are more powerful than one agent with $O((\log n)^{1-\epsilon})$ bits of memory and p pebbles. Our results again allow to fully describe the tradeoff between the number of agents and the memory of each agent. When agents have $\Omega(\log n)$ memory, a single agent explores all n-vertex graphs. For agents with $O((\log n)^{1-\epsilon})$ memory, $\Omega(\log\log n)$ agents are needed. However, when $\Omega(\log\log n)$ agents are available it is sufficient that each of them has only constant memory.

1.2 Related Work

Exploration of mazes. Exploration algorithms were first designed for mazes, which are finite, connected, and anonymous subgraphs of the two-dimensional grid where edges are labeled with their cardinal direction. Shannon [38] constructed an actual physical device—Shannon's mouse—that explores a 5×5 grid and uses constant memory per vertex. Budach [9] gave a proof that for every agent with finite memory there is a maze that the agent cannot explore. Shah [37] showed that an agent with finite memory and five pebbles can explore all mazes. Subsequently, the number of pebbles needed for the exploration of all mazes has been reduced to four by Blum and Sakoda [8] and to two by Blum and Kozen [7]. Hoffmann [25] proved that for any finite memory agent with one pebble there is a maze that the agent cannot explore, implying that two is the minimum number of pebbles for a finite agent to explore all finite mazes. Blum and Kozen [7] also showed that all mazes can be explored by two agents with finite memory.

Exploration of undirected graphs. Blum and Kozen [7] studied a generalization of the maze exploration problem to arbitrary finite, undirected, and planar graphs with constant maximal degree. To allow for sensible decisions of the agents, they assume that for each vertex the outgoing edges have a locally unique outgoing port number and the agents transition based on this information. They showed that three finite memory agents cannot explore all such graphs when their transition depends on the last edge taken, the states of all agents that took the same edge, and the set of edges at the current vertex. Rollik [34] strengthened this result showing that for any finite set of agents with finite memory there is a planar graph—a so-called *trap*—that cannot be explored. For k agents, the trap is of order $\tilde{O}(s \uparrow\uparrow (2k+1))$, where $a \uparrow\uparrow b := a^{a^{\cdot^{\cdot^{a}}}}$ with b levels in the exponent. This bound was improved by Fraigniaud et al. [24] to $\tilde{O}(s \uparrow\uparrow (k+1))$, and is further improved in this article (see Section 1.1.2).

Aleliunas et al. [2] showed that a random walk of length $n^5 \log n$ explores any n-vertex graph with high probability. By the probabilistic method, this implies the existence of a universal traversal sequence (UTS) of polynomial length for regular graphs. A (n, d)-UTS is a sequence of port numbers in $\{0, \dots, d-1\}$ such that an agent starting in an arbitrary vertex and following the port numbers of the sequence explores every d-regular graph with n vertices. Using a counter for the number of steps, this yields a log-space randomized algorithm constructing a UTS when the number of vertices is known. Using Nisan's derandomization technique [32], this yields a deterministic algorithm with $O(\log^2 n)$ memory. The length of the sequence, however, increases to $2^{O(\log^2 n)} = O(n^{\log n})$. Explicit constructions of UTS are only known for cycles (Istrail [26]) and to date it remains open whether a UTS of polynomial length can be constructed deterministically in log-space for general graphs.

As a remedy for the perceived difficulty of constructing a UTS, Koucký [28] introduced the related concept of universal exploration sequences (UXS) where the next port number depends on the port number of the edge to the previously visited vertex. Reingold [33] showed that an (n, d)-UXS can be constructed deterministically in $O(\log n)$ space when n is known. A slight modification of his algorithm allows us to explore any (not necessarily) regular graph, thus, solving the undirected s-t connectivity problem in log-space. Fraigniaud et al. [22] showed that $\Omega(\log n)$ memory is necessary to explore all anonymous undirected graphs with up to n vertices. The set of graphs $\mathcal{G}_k$ that can be explored by an agent with k states is increasing in the sense that there is polynomial $h : \mathbb{N} \to \mathbb{N}$ such that $\mathcal{G}_{h(i)}$ is strictly included in $\mathcal{G}_i$; see Fraigniaud et al. [23].

Koucký [28] noted that 1^{2n} is a UXS for trees. As remarked by Diks et al. [16], this gives rise to a *perpetual* tree exploration algorithm that runs forever, and eventually visits all vertices of the tree. For the case that the exploration algorithm is required to terminate they showed that $\Omega(\log \log \log n)$ bits of memory are needed. If the algorithm is even required to terminate at the same vertex where it started, then $\Omega(\log n)$ bits of memory are needed. A matching upper bound for the latter problem was given by Ambühl et al. [3].

There are further results on the exploration of undirected anonymous edge-labeled graphs by agents with unconstrained memory. Dudek et al. [18] showed that an agent provided with a pebble can explore (and even map) an undirected graph in time $O(mn)$, where m is the number of edges. Chalopin et al. [12] showed that if the starting node can be recognized by the agent, the agent can explore the graph in time $O(n^3\Delta)$ using $O(n\Delta \log n)$ memory. Exploration with the objective of minimizing the exploration time also has been studied in terms of competitive analysis. In this context an exploration algorithm for an edge weighted graph is called c-competitive if the sum of the weights of the edges traversed by the algorithm is at most c times that of an optimal offline walk. If the task is to visit all vertices of a vertex-labeled weighted graph and an agent learns about all neighbors when arriving at a node, then a nearest neighbor greedy approach is $\Theta(\log n)$-competitive [35], and there is a $\Theta(\log n)$-competitive hierarchical-DFS approach [30]. Algorithms with constant competitive ratios are known for unweighted graphs and cycles [31], as well as graphs with bounded genus [27, 30].

Exploration of directed graphs. Exploration is considerably more difficult for directed graphs, e.g., random walks may need exponential time to visit all vertices. Without any constraints on memory, Bender et al. [4] gave an $O(n^8\Delta^2)$-time algorithm that uses one pebble and explores (and maps) an unlabeled directed graph with maximum degree Δ, when an upper bound on the number n of vertices is known. For the case that such an upper bound is not available, they proved that $\Theta(\log\log n)$ pebbles are both necessary and sufficient. Concerning the space complexity of directed graph exploration, Fraigniaud and Ilcinkas [21] showed that $\Omega(n \log \Delta)$ bits of memory are necessary to explore any directed graph with n vertices and maximum degree Δ, even with a linear number of pebbles. They provided an $O(n\Delta \log n)$-space algorithm for terminating exploration with an exponential running time using a single pebble. They also gave an $O(n^2\Delta \log n)$-space algorithm running in polynomial time and using $O(\log\log n)$ indistinguishable pebbles. According to Bender et al. [4] at least $\Omega(\log\log n)$ pebbles are necessary in this setting. For the exploration time of labeled directed graphs, where the task is to traverse all edges, the competitive ratios achievable by online algorithms are closely related to the deficiency of the graph [1, 15, 19]. If, however, the agent learns about all neighbors when arriving at a node and has to visit all vertices of the graph, then the best possible competitive ratio is $\Theta(n)$, even for Euclidean planar graphs [20].

Further exploration results. Further related is the problem of exploring geometric structures in the plane; see, e.g., References [10, 11] for the exploration of simple polygons and References [6, 14] for the exploration of other geometric terrains.

1.3 Techniques and Outline of the Paper

In Section 2, we fix notation and introduce the agent models considered in this article. We show that for undirected graph exploration a pebble is more powerful than a bit of memory (Lemma 2.1). We further show that under some additional assumptions two additional agents are more powerful than a pebble (Lemma 2.3).

Our main positive result is presented in Section 3, where we give a single agent exploration algorithm that explores any n-vertex graph with $O(\log\log n)$ pebbles and $O(\log\log n)$ bits of memory. In light of the lemmas presented in Section 2, we obtain as direct corollaries that (a) a single agent with $O(\log\log n)$ pebbles and constant memory can explore any n-vertex graph and (b) a set of $O(\log\log n)$ agents with constant memory each can explore any n-vertex graph.

For the algorithm, we use the concept of universal exploration sequences due to Koucký [28]. One of our main building blocks is the algorithm of Reingold [33] that takes n and d as input and deterministically constructs an exploration sequence universal for all d-regular graphs using $O(\log n)$ bits of memory. The general idea of our algorithm is to run Reingold's algorithm with a smaller amount of seed memory a. As the seed memory is substantially less than $O(\log n)$, the algorithm will, in general, fail to explore the whole graph. We show, however, that the algorithm will visit $2^{\Omega(a)}$ distinct vertices (Lemma 3.3). Reinvoking Reingold's algorithm allows us to *deterministically* walk along these vertices in the order of exploration of Reingold's algorithm. Using this traversal, we encode additional memory by placing a subset of pebbles on the vertices along the walk. Having boosted our memory this way, we again run Reingold's algorithm, this time with more memory, and recurse. At some recursion depth, running Reingold's algorithm with a^* bits of memory will visit less than $2^{\Omega(a^*)}$ distinct vertices. We show that this can only happen when the graph is fully explored, which allows us to terminate the algorithm when this event occurs. Unwrapping all levels of recursion then also allows us to return to the starting vertex. The ability of our algorithm to terminate and return to the starting vertex after successful exploration, stands in contrast to Reingold's algorithm that is only able to terminate when being given the number n of vertices as input.

There are a couple of technical difficulties to make these ideas work. The main challenge is that the memory generated by placing pebbles along a walk in the graph is implicit and can only be accessed and altered locally. To still make use of the memory, we do not work with Reingold's algorithm directly but consider an implementation of Reingold's algorithm on a Turing machine with logarithmically bounded working tape. We show that the tape operations on the working tape can be reproduced by the agent by placing and retrieving the pebbles on the walk. This allows us to use the memory encoded by the pebble positions for further runs of Reingold's algorithm. In each recursion, we only need a constant number of pebbles and additional states. We further show that $O(\log \log n)$ recursive calls are necessary to explore an n-vertex graph so that the total number of pebbles needed is $O(\log \log n)$.

A second challenge is that Reingold's algorithm produces a UXS for regular graphs, which our graph need not be. A natural approach to circumvent this issue is to apply the technique of Koucký [29] that allows us to locally view vertices with degree d as cycles of $3d$ subvertices with degree 3 each. Unfortunately, this approach requires $\Theta(\log d)$ bits of memory if we keep track of the current subvertex, which may exceed the memory of our agent. To circumvent this issue, we store the current subvertex only implicitly and navigate the graph in terms of subvertex index offsets instead of the actual subvertex indices.

Our general lower bound is presented in Section 4. Specifically, we show that for a set of cooperative agents with sublogarithmic memory of $O((\log n)^{1-\varepsilon})$ for some constant $\varepsilon > 0$, $\Omega(\log \log n)$ agents are needed to explore any undirected graph with n vertices. The same lower bound construction shows that an agent with sublogarithmic memory needs $\Omega(\log \log n)$ pebbles to explore any n-vertex graph.

To prove the lower bound, we use the concept of an r-barrier. Informally, an r-barrier is a graph with two special entry points such that any subset of up to r agents with s states cannot reach one entry point when starting from the other. Moreover, a set of $r + 1$ agents can explore an r-barrier, but the agents can only leave the barrier via the same entry point. We recursively construct an r-barrier by replacing edges by $(r - 1)$-barriers. A set of r agents traversing this graph needs to stay close to each other to be able to traverse the barriers and make progress. However, if the agents stay close to each other, the states and relative positions of the agents become periodic relatively quickly, and we can use this fact to build an r-barrier. By carefully bounding the size of the r-barriers in our recursive construction, we obtain a trap of size $O(s^{2^{5k}})$ for any given set of k agents with at most s states each. The size of the trap directly implies that the number of agents with at most $O((\log n)^{1-\varepsilon})$ bits of memory needed for exploring any graph of size n is at least $\Omega(\log \log n)$.

2 TERMINOLOGY AND MODEL

2.1 The Graph

Let $G = (V, E)$ be an undirected graph with n vertices to be explored by a single agent or a group of agents. All agents start at the same vertex $v_0 \in V$. We say that G is explored when each vertex of G has been visited by at least one agent. Every graph considered in this article is assumed to be connected as otherwise exploration is not possible. We further assume that the graph is anonymous, i.e., the vertices of the graph are unlabeled and therefore cannot be identified or distinguished by the agents. To enable sensible navigation, the edges incident to a vertex v have distinct labels $0, \ldots, d_v - 1$, where d_v is the degree of v. This way every edge $\{u, v\}$ has two—not necessarily identical—labels called *port numbers*, one at u and one at v.

2.2 The Agents

We model an agent as a tuple $A = (\Sigma, \bar{\Sigma}, \delta, \sigma^*)$, where Σ is its set of states, $\bar{\Sigma} \subseteq \Sigma$ is its set of halting states, $\sigma^* \in \Sigma$ is its starting state, and δ is its transition function. The transition function governs the actions of the agent and its transitions between states based on its local observations. Its exact specifics depend on the problem considered, i.e., whether we consider a single agent or a group of agents and whether we allow the agents to use pebbles. Exploration terminates when a halting state is reached by all agents.

2.2.1 A Single Agent Without Pebbles. The most basic model is that of a single agent A without any pebbles. In each step, the agent observes its current state $\sigma \in \Sigma$, the degree d_v of the current vertex v and the port number l at v of the edge from which v was entered. The port number l is also referred to as *incoming port number*. We let $l = \perp$ at the start of the exploration or when the agent stayed at v in the last transition. The transition function δ then specifies a new state $\sigma' \in \Sigma$ of the agent and an *outgoing port number* $l' \in \{0, \ldots, d_v - 1\} \cup \{\perp\}$. If $l' \in \{0, \ldots, d_v - 1\}$, then the agent enters the edge with the local port number l', whereas for $l' = \perp$ it stays at v. Formally, the transition function is a partial function:

$$\begin{aligned} \delta : \Sigma \times \mathbb{N} \times (\mathbb{N} \cup \{\perp\}) &\to \Sigma \times (\mathbb{N} \cup \{\perp\}), \\ (\sigma, d_v, l) &\mapsto (\sigma', l'). \end{aligned}$$

Note that the transition function only needs to be defined for l with $l < d_v$ and degrees d_v that actually appear in the class of graphs considered. It is standard to define the space requirement of an an agent with states Σ as $\log |\Sigma|$ as this is the number of bits needed to encode every state; see, e.g., Cook and Rackoff [13].

2.2.2 A Single Agent with Pebbles. We may equip the agent A with a set $P = \{1, \ldots, p\}$ of unique and distinguishable pebbles. At the start of the exploration the agent is carrying all of its pebbles. As before, the agent observes in each step the degree d_v of the current vertex v and the port number l from which v was entered, allowing for $l = \perp$ in case the agent did not move during the previous transition. In addition, the agent has the ability to observe the set of pebbles P_A that it carries and the set of pebbles P_v present at the current vertex v. The transition function δ then specifies the new state $\sigma' \in \Sigma$ of the agent, and a move $l' \in \{0, \ldots, d_v - 1\} \cup \{\perp\}$ as before. In addition, the agent may drop any subset $P_{\text{drop}} \subseteq P_A$ of carried pebbles and pick up any subset of pebbles $P_{\text{pick}} \subseteq P_v$ that were located at v, so that after the transition the set of carried pebbles is $P'_A = (P_A \setminus P_{\text{drop}}) \cup P_{\text{pick}}$ and the set of pebbles present at v is $P'_v = (P_v \setminus P_{\text{pick}}) \cup P_{\text{drop}}$. Formally, we have

$$\delta : \Sigma \times \mathbb{N} \times (\mathbb{N} \cup \{\perp\}) \times 2^P \times 2^P \to \Sigma \times (\mathbb{N} \cup \{\perp\}) \times 2^P \times 2^P,$$
$$(\sigma, d_v, l, P_A, P_v) \mapsto (\sigma', l', P_A', P_v').$$

The transition function δ is partial as it is only defined for $P_A \cap P_v = \emptyset$. We assume that the pebbles are actual physical devices dropped at the vertices so that no space is needed to manage the pebbles. Thus, the space requirement of the agent is again $\log|\Sigma|$.

2.2.3 A Set of Agents without Pebbles. Consider a set of k cooperative agents $A_1 = (\Sigma_1, \bar{\Sigma}_1, \delta_1, \sigma_1^*), \dots, A_k = (\Sigma_k, \bar{\Sigma}_k, \delta_k, \sigma_k^*)$ jointly exploring the graph. We assume that all agents start at the same vertex v_0. In each step, all agents synchronously determine the set of agents they share a location with, as well as the states of these agents. Then, all agents move and alter their states synchronously according to their transition functions $\delta_1, \dots, \delta_k$. The transition function of agent i determines a new state σ' and a move l' as before. Formally, let

$$\Sigma_{-i} = (\Sigma_1 \cup \{\perp\}) \times \cdots \times (\Sigma_{i-1} \cup \{\perp\}) \times (\Sigma_{i+1} \cup \{\perp\}) \times \cdots \times (\Sigma_k \cup \{\perp\})$$

denote the states of all agents potentially visible to agent A_i where a $\perp$ at position j (or $(j-1)$ if $j \geq i$) stands for the event that agent A_i and agent A_j are located at different vertices. Then, the transition function δ_i of agent A_i is a partial function:

$$\delta_i : \Sigma_i \times \Sigma_{-i} \times \mathbb{N} \times (\mathbb{N} \cup \{\perp\}) \to \Sigma_i \times (\mathbb{N} \cup \{\perp\}),$$
$$(\sigma_i, \boldsymbol{\sigma}_{-i}, d_v, l) \mapsto (\sigma_i', l_i').$$

The overall memory requirement is $\sum_{i=1}^{k} \log|\Sigma_i|$.

2.3 Relationship between Agent Models

To compare the capability of an agent A with s states and p pebbles to another agent A' with s' states and p' pebbles or a set of agents $\mathcal{A}$, we use the following notion: We say that the walk of an agent A is *reproduced* by an agent A' in a graph G, if the sequence of edges traversed by A is a subsequence of the edges visited by A' in G, and agent A' reaches a halting state if and only if agent A reaches a halting state. Put differently, A traverses the same edges as A' in the same order, but for every edge traversal of A the agent A' can do an arbitrary number of intermediate edge traversals. Similarly, we say that a set of agents $\mathcal{A}$ reproduces the walk of an agent A in G, if there is an agent $A' \in \mathcal{A}$ such that A' reproduces the walk of A in G. We further require that if agent A reaches a halting state then all agents $A' \in \mathcal{A}$ reach a halting state.

We first formally show the intuitive fact that pebbles are more powerful than memory bits.

Lemma 2.1. *Let A be an agent with s states and p pebbles exploring a set of graphs $\mathcal{G}$. Then there is an agent A' with five states and $p + \lceil \log s \rceil$ pebbles that reproduces the walk of A in every $G \in \mathcal{G}$ and performs at most three edge traversals for every edge traversal of A.*

Proof. As the set of graphs $\mathcal{G}$ that can be explored by an agent with s states and p pebbles is non-decreasing in s, it suffices to show the claimed result for the case that s is an integer power of two. Let $A = (\Sigma, \bar{\Sigma}, \delta, \sigma^*)$ be an agent with a set of p pebbles P and $s = |\Sigma| = 2^r$, $r \in \mathbb{N}$ states exploring all graphs $G \in \mathcal{G}$. In the following, we construct an agent $A' = (\Sigma', \bar{\Sigma}', \delta', \sigma^{*\prime})$ with five states $\Sigma' = \{\sigma^{*\prime}, \sigma_{\text{comp}}, \bar{\sigma}_{\text{halt}}, \sigma_{\text{back}-1}, \sigma_{\text{back}-2}\}$, one halting state $\bar{\Sigma}' = \{\bar{\sigma}_{\text{halt}}\}$, and a set P' of $|P'| = p + r$ pebbles. The general idea is to let A' store the state of A by dropping and retrieving the additional r pebbles. To this end, we identify p of the pebbles of A' with the p pebbles of A and call the additional set of r pebbles P_Σ', i.e., $P' = P \cup P_\Sigma'$ with $|P| = p$ and $|P_\Sigma'| = r$, respectively. Since $|P_\Sigma'| = r$ and $|\Sigma| = s = 2^r$, there is a canonical bijection $f : \Sigma \to 2^{P_\Sigma'}$. The construction ensures that the following invariant holds during the traversal: If agent A reaches a vertex v in a state σ, then agent A' reaches v in its computation state σ_{comp} while carrying the set of pebbles $f(\sigma)$ plus the additional pebbles that A is carrying and all pebbles in $P'_\Sigma \setminus f(\sigma)$ are located at v. We need the additional states $\sigma_{\text{back}-1}$ and $\sigma_{\text{back}-2}$ to move all pebbles in P'_Σ encoding the state of A to the next vertex in some intermediate steps.

For every $P_{A'} \subseteq P'$, $P_v \subseteq P'$ with $P_{A'} \cap P_v = \emptyset$, $d_v \in \mathbb{N}$, $l \in \mathbb{N} \cup \{\perp\}$ and $\sigma \in \Sigma$, let

$$\delta(\sigma, d_v, l, P_{A'} \cap P, P_v \cap P) = (\sigma', l', P'_A, P'_v) \tag{1}$$

be the transition of agent A with $\sigma' \in \Sigma$, $l' \in \mathbb{N} \cup \{\perp\}$ and $P'_A, P'_v \in 2^P$. Then, we define

$$\delta'(\sigma_{A'}, d_v, l, P_{A'}, P_v) = \begin{cases} (\sigma_{\text{comp}}, \ \ l', P'_A \cup f(\sigma'), P'_v \cup (P'_\Sigma \setminus f(\sigma')) & \text{if } l' = \perp \text{ and } \sigma' \notin \bar{\Sigma}, \\ (\sigma_{\text{back}-1}, l', P'_A \cup (P'_\Sigma \setminus f(\sigma')), P'_v \cup f(\sigma')) & \text{if } l' \neq \perp \text{ and } \sigma' \notin \bar{\Sigma}, \\ (\bar{\sigma}_{\text{halt}}, \ \ l', P'_A \cup f(\sigma'), P'_v \cup (P'_\Sigma \setminus f(\sigma')) & \text{else,} \end{cases} \tag{2}$$

for $\sigma_{A'} \in \{\sigma^{*\prime}, \sigma_{\text{comp}}\}$. If agent A traverses an edge without entering a halting state (second case in transition function δ' above), then we also need to fetch the set of remaining pebbles $f(\sigma')$ from the previous vertex to be able to encode the state of A in the future. To this end, A' switches to the state $\sigma_{\text{back}-1}$. When in state $\sigma_{\text{back}-1}$, the fetching will be done in two steps: First, A' drops all pebbles in $P'_\Sigma \setminus f(\sigma')$, moves to the previous vertex and changes its state to $\sigma_{\text{back}-2}$. Formally, this means

$$\delta'(\sigma_{\text{back}-1}, d_v, l, P_{A'}, P_v) = (\sigma_{\text{back}-2}, l, P_{A'} \setminus P'_\Sigma, P_v \cup (P'_\Sigma \cap P_{A'}))$$

for all $d_v \in \mathbb{N}$, $l \in \mathbb{N} \cup \{\perp\}$ and $P_{A'}, P_v \in 2^{P'}$ with $P_{A'} \cap P_v = \emptyset$. Then it picks up the pebbles in $f(\sigma')$, returns to the current vertex of A and changes its state to σ_{comp}, i.e.,

$$\delta'(\sigma_{\text{back}-2}, d_v, l, P_{A'}, P_v) = \left(\sigma_{\text{comp}}, l, P_{A'} \cup (P'_\Sigma \cap P_v), P_v \setminus P'_\Sigma\right)$$

for all $d_v \in \mathbb{N}$, $l \in \mathbb{N} \cup \{\perp\}$ and $P_{A'}, P_v \in 2^{P'}$ with $P_{A'} \cap P_v = \emptyset$. After these two transitions, the state of agent A' is σ_{comp}, all pebbles in P'_Σ are at the current vertex or carried by A' and $P_{A'} \cap P'_\Sigma = f(\sigma')$ encodes the current state σ' of agent A.

A simple inductive proof establishes that the state σ of A in every step of the exploration of a graph $G \in \mathcal{G}$ corresponds to the set of pebbles in P'_Σ carried by A' in its computation state σ_{comp}, i.e., $\sigma = f^{-1}(P_{A'} \cap P'_\Sigma)$. Moreover, if agent A in state σ traverses an edge $\{v, w\}$ from a vertex v to a vertex w and does not move to a halting state, then A' will traverse the edge $\{v, w\}$ three times and afterwards again the set of pebbles carried by A will correspond to $P_{A'} \cap P$ and the state of A to $\sigma = f^{-1}(P_{A'} \cap P'_\Sigma)$. If A remains at the same vertex or moves to a halting state, then this transition is mirrored by a single transition of agent A'. In particular, agent A' visits exactly the same vertices as A in every graph $G \in \mathcal{G}$ while performing at most three times the number of edge traversals. □

We proceed to show a similar reduction between exploration with additional agents and exploration by a single agent with pebbles. Intuitively, it would seem that an additional agent is at least as powerful as a pebble, since an agent may simply simulate the behavior of a pebble. However, there are several subtleties in the different behavior of agents and pebbles that prevent us from showing such a general result. While a pebble is passive and its movement is entirely determined by the agent, an additional agent moves on its own and has to compute where to go next. In addition, after termination of the exploration, pebbles may remain distributed in the graph while agents that mimic pebbles need to be informed about the fact that the exploration terminated to switch into a halting state.

We resolve these issues in the following way. First, we restrict ourselves to agents with pebbles that have the additional property that they carry all pebbles when they terminate. Second, we require that the agent with pebbles satisfies the following pickup invariant. Our algorithm presented in Section 3 will satisfy both properties.

Definition 2.2. Let A be an agent with a set of p pebbles P. We say that A satisfies the *pickup invariant*, if every time A drops a pebble $p_0 \in P$ at a vertex v with current incoming edge label l, then p_0 is only picked up again if A is at v with the same incoming edge label l.

For illustration of the general proof idea assume for a moment that agents are able to recall their last incoming port number even when they stay at a vertex for multiple rounds. Then, an agent A with p pebbles satisfying the pickup invariant can be simulated by a set of $p + 1$ agents as follows. There is one master agent reproducing the walk of A while all other agents behave and move as pebbles. When a pebble agent remains at a vertex it does not move until the master agent visits the vertex again. Using the pickup invariant and the fact that the pebble agent can recall the last incoming port number, the pebble agent can do the same computations as the master agent and can synchronously move as the master agent in case the outcome of the computation is that the corresponding pebble is picked up by agent A.

For this to work, we need a way to circumvent the fact that the incoming port number of an agent becomes $\perp$ whenever they remain at a vertex. Instead of remaining at the vertex v where the agent with pebbles would drop a pebble, we let the corresponding pebble agents move back and forth between v and the vertex v' last visited by the agent before entering v. This ensures that in every other step, the pebble agent is where it is supposed to be. Since the master agent may be using cycles of odd length, we need to ensure that the agent knows of the position of the pebbles when a pebble agent is not at the vertex where the pebble was dropped. To this end, we double the number of agents and do two (largely independent) explorations of the graph such that the second exploration is always one step behind the first exploration. That is, there are two master agents each with a set of p pebble agents. The master agents do not distinguish between their pebble agents and the pebble agents of the other master agent. Since the explorations are shifted by one step, the two pebble agents of the two master agents simulating the same pebble will not be at the same vertex so that the agent not picked up by one master agent will be picked up by the other. Formally, we show the following result.

Lemma 2.3. *Let A be an agent with s states and p pebbles that explores a set $\mathcal{G}$ of graphs, satisfies the pickup invariant and terminates carrying all pebbles. Then there is a set $\mathcal{A}$ of $2(p + 1 + \lceil \log s \rceil)$ agents with a constant number of states each that reproduce the walk of A in every graph $G \in \mathcal{G}$. If A never remains at a vertex before halting, then for every edge traversal of A, each agent in $\mathcal{A}$ performs at most one edge traversal.*

Proof. For an agent with pebbles, there is no benefit in remaining at a vertex so that there is no loss of generality when assuming for the following arguments that A never remains at a vertex before halting.

Let $A = (\Sigma, \bar{\Sigma}, \delta, \sigma^*)$ be an agent with $s = |\Sigma|$ and a set $P = \{1, \ldots, p\}$ of p pebbles exploring all graphs $G \in \mathcal{G}$. We first explain the construction for the case that s is constant and describe at the end of the proof how to use a construction similar to the proof of Lemma 2.1 when s is not constant. We construct a set $\mathcal{A} = \{A_{1,0}, \ldots, A_{1,p}, A_{2,0}, \ldots, A_{2,p}\}$ of $2p + 2$ agents $A_{j,i} = (\Sigma_{j,i}, \bar{\Sigma}_{j,i}, \delta_{j,i}, \sigma^*_{j,i})$, $i \in \{0, \ldots, p\}$, $j \in \{1, 2\}$ that reproduces the walk of A on all graphs $G \in \mathcal{G}$. In this construction, there are two explorations by the groups of agents $A_{1,0}, \ldots, A_{1,p}$ and $A_{2,0}, \ldots, A_{2,p}$, where in each group $j \in \{1, 2\}$ the master agent $A_{j,0}$ represents the original agent A while every agent $A_{j,i}$ for

$i > 0$ represents a pebble. In the course of the exploration, the assignment of pebble agents to the two explorations may change, however.

For the first master agent $A_{1,0}$, we set $\Sigma_{1,0} = \Sigma$, $\bar{\Sigma}_{1,0} = \bar{\Sigma}$, and $\sigma^*_{1,0} = \sigma^*$. The second master agent has one additional state σ_{start} that allows it to wait at the start of the exploration for one step, i.e., we set $\Sigma_{2,0} = \Sigma \cup \{\sigma_{\text{start}}\}$, $\bar{\Sigma}_{2,0} = \bar{\Sigma}$ and $\sigma^*_{2,0} = \sigma_{\text{start}}$. Waiting for one time unit is implemented via the transition

$$\delta_{2,0}(\sigma_{\text{start}}, \boldsymbol{\sigma}_{-2,0}, d_v, \perp) = (\sigma^*, \perp)$$

for all $d_v \in \mathbb{N}$ and $\boldsymbol{\sigma}_{-2,0} \in \Sigma_{-2,0}$. Throughout the construction, we will ensure that the exploration by master agent $A_{2,0}$ is always one step behind the exploration of master agent $A_{1,0}$. In addition, master agent $A_{1,0}$ explores all graphs in the same way as the agent with pebbles A would. Since we assumed that A never remains at a vertex, this implies in particular that $A_{1,0}$ and $A_{2,0}$ are never at the same vertex (except for the start and the end of the exploration).

For every agent $A_{j,i}$ with $i \in P$ and $j \in \{1,2\}$, we set $\Sigma_{j,i} = \{c_{j,i}, d_{j,i}, d'_{j,i}, \bar{h}_{j,i}\}$, $\bar{\Sigma}_{j,i} = \{\bar{h}_{j,i}\}$. Intuitively, the state $c_{j,i}$ simulates that pebble i is carried, $d_{j,i}$ simulates that the pebble is dropped and at the correct vertex, $d'_{j,i}$ simulates that the pebble is dropped but at a neighbor of the correct vertex, and $\bar{h}_{i,j}$ is the halting state. We let pebble agents of the first group start being carried and the pebble agents of the second group start in the incorrect dropped state, i.e., $\sigma^*_{1,i} = c_{1,i}$ and $\sigma^*_{2,i} = d'_{2,i}$ for all $i \in P$. For $i, i' \in \{0, \dots, p\}$ and $j, j' \in \{1,2\}$, let $\sigma_{j',i':j,i}$ denote the state of agent $A_{j',i'}$ visible to agent $A_{j,i}$, i.e., $\sigma_{j',i':j,i} = \sigma_{j,i}$ if $A_{j',i'}$ and $A_{j,i}$ share the same vertex and $\sigma_{j',i':j,i} = \perp$ otherwise. To define the transition functions $\delta_{j,i}(\sigma_{j,i}, \boldsymbol{\sigma}_{-j,i}, d_v, l)$ for $i \in \{0, \dots, p\}$, $\sigma_{j,i} \in \Sigma_{j,i}$, $\boldsymbol{\sigma}_{-j,i} \in \Sigma_{-j,i}$ and $d_v \in \mathbb{N}$, $l \in \mathbb{N} \cup \{\perp\}$, we compute for all $j \in \{1,2\}$ the corresponding transitions of the pebble agent A, i.e.,

$$\delta(\sigma_{j,0}, d_v, l, C(\boldsymbol{\sigma}_{-j,0}), D(\boldsymbol{\sigma}_{-j,0})) = (\sigma'_{j,0}, l'_j, P'_{j,A}, P'_{j,v}),$$

with $\sigma'_{j,0} \in \Sigma$, $l'_j \in \mathbb{N}$, and $P'_{j,A}, P'_{j,v} \in 2^P$, where the set

$$C(\boldsymbol{\sigma}_{-j,0}) := \{i \in P : \sigma_{1,i:j,0} = c_{1,i}\} \cup \{i \in P : \sigma_{2,i:j,0} = c_{2,i}\}$$

is the set of pebbles that are simulated by a pebble agent (of the first or the second group) who is at the same vertex as the master agent $A_{j,0}$ and in its carried state. Similarly, the set

$$D(\boldsymbol{\sigma}_{-j,0}) := \{i \in P : \sigma_{1,i:j,0} = d_{1,i}\} \cup \{i \in P : \sigma_{2,i:j,0} = d_{2,i}\}$$

is the set of pebbles that are simulated by a pebble agent (of the first or the second group) who is at the same vertex as the master agent $A_{j,0}$ and in its correct dropped state $d_{1,i}$ or $d_{2,i}$. We then let the master agents $A_{1,0}$ and $A_{2,0}$ transition as agent A when observing the corresponding sets of carried and dropped pebbles, i.e.,

$$\delta_{j,0}(\sigma_{j,0}, \boldsymbol{\sigma}_{-j,0}, d_v, l) = (\sigma'_{j,0}, l'_j)$$

for all $j \in \{1,2\}$. Pebble agents that are in their carried state move as the master agent that carries them. Except for the start of the exploration, there is at most one master agent at a vertex (and pebble agents of the second group start dropped) so that the assignment to master agents is well-defined. Pebble agents $A_{j,i}$ that are in their correct dropped state $d_{j,i}$ or the carried state $c_{j,i}$ observe whether there is a master agent at the same vertex. If this is the case, then they do the same computation as the master agent. If the outcome of this computation is that the corresponding pebble is picked up, then they move as the master agent and transition into their carried state. If the outcome of this computation is that the pebble should remain dropped, then they move to the last incoming port number and transition into the incorrect dropped state $d'_{j,i}$. Pebble agents that are in the incorrect dropped state $d_{j,i}$ move to the last incoming port number and transition into the correct dropped state $d_{j,i}$. There is a further small corner case of pebble agents of the second group that transition from dropped to carried at the start of the exploration. Summarizing, we obtain

$$\delta_{j,i}(\sigma_{j,i},\boldsymbol{\sigma}_{-j,i},d_v,l)=\begin{cases}(d_{j,i},l) & \text{if } \sigma_{j,i}=d'_{j,i} \text{ and } \sigma_{2,0:j,i}\neq\sigma_{\text{start}},\\ (c_{j,i},\perp) & \text{if } \sigma_{j,i}=d'_{j,i} \text{ and } \sigma_{2,0:j,i}=\sigma_{\text{start}},\\ (d'_{j,i},l) & \text{if } \sigma_{j,i}=d_{j,i} \text{ and } \sigma_{1,0:j,i}=\sigma_{2,0:j,i}=\perp,\\ (c_{j,i},l'_1) & \text{if } \sigma_{j,i}\in\{c_{j,i},d_{j,i}\},\ \sigma_{1,0:j,i}\neq\perp,\ \sigma'_{1,0}\notin\bar{\Sigma}_{1,0}, \text{ and } j\in P'_{1,A},\\ (d'_{j,i},l) & \text{if } \sigma_{j,i}\in\{c_{j,i},d_{j,i}\},\ \sigma_{1,0:j,i}\neq\perp,\ \sigma'_{1,0}\notin\bar{\Sigma}_{1,0}, \text{ and } j\in P'_{1,v},\\ (\bar{h}_{j,i},l'_1) & \text{if } \sigma_{j,i}\in\{c_{j,i},d_{j,i}\},\ \sigma_{1,0:j,i}\neq\perp, \text{ and } \sigma'_{1,0}\in\bar{\Sigma}_{1,0},\\ (c_{j,i},l'_2) & \text{if } \sigma_{j,i}\in\{c_{j,i},d_{j,i}\},\ \sigma_{2,0:j,i}\notin\{\perp,\sigma_{\text{start}}\},\ \sigma'_{2,0}\notin\bar{\Sigma}_{2,0}, \text{ and } j\in P'_{2,A},\\ (d'_{j,i},l) & \text{if } \sigma_{j,i}\in\{c_{j,i},d_{j,i}\},\ \sigma_{2,0:j,i}\notin\{\perp,\sigma_{\text{start}}\},\ \sigma'_{2,0}\notin\bar{\Sigma}_{2,0}, \text{ and } j\in P'_{2,v},\\ (\bar{h}_{j,i},l'_2) & \text{if } \sigma_{j,i}\in\{c_{j,i},d_{j,i}\},\ \sigma_{2,0:j,i}\notin\{\perp,\sigma_{\text{start}}\}, \text{ and } \sigma'_{2,0}\in\bar{\Sigma}_{2,0},\end{cases}$$

for all $i \in P$ and $j \in \{1, 2\}$.

To finish the proof, fix a graph $G \in \mathcal{G}$ and consider the transitions of agent A and the set of agents $\mathcal{A}$ in G. An inductive proof shows that after k transitions, the state and position of agent A equals the state and position of the master agent $A_{1,0}$, and the state and position of the master agent $A_{2,0}$ equals the state and position of A after $k-1$ transitions. In addition, there is exactly one pebble agent $A_i \in \{A_{1,i}, A_{2,i}\}$ that is in state $c_{j,i}$ with $j \in \{1, 2\}$ at the same vertex as master agent $A_{1,0}$ if and only if agent A is carrying pebble i. For the dropped pebbles one can show that there is exactly one pebble agent $A_i \in \{A_{1,i}, A_{2,i}\}$ that is in state $d_{j,i}$ with $j \in \{1, 2\}$ at some vertex v if and only if in the exploration with pebbles agent A has dropped (and not yet retrieved) pebble i at vertex v an even number of steps ago. So when master agent $A_{1,0}$ returns to a vertex where A dropped a pebble it either sees the pebble agent that it caused to go into dropped state (if since that an even number of steps have passed), or the pebble agent that the other master agent $A_{2,0}$ caused to go into dropped state (if an odd number of steps have passed). In any way, the two explorations by the master agents $A_{1,0}$ and $A_{2,0}$ are never at the same vertex at any point in time, and the dropped pebbles seen by the one are invisible to the other, so that the claim follows. Finally, note that as agent A carries all pebbles at the end of the exploration, all pebbles agents are at the same vertex as a master agent when they switch to a halting state. Thus, also all pebble agents reach a halting state when the exploration is terminated.

Finally, when s is not constant, we introduce $2k = 2\lceil \log s \rceil$ additional agents $A_{1,p+1}, \ldots, A_{1,p+k}$ and $A_{2,p+1}, \ldots, A_{2,p+k}$ with states $\Sigma_{j,i} = \{0, 1, \bar{h}\}$ and halting states $\bar{\Sigma} = \{\bar{h}\}$ for all $j \in \{1, 2\}$ and $i \in \{p+1, p+k\}$. The general idea is that the additional agents $\mathcal{A}'_1 = \{A_{1,i} : i \in \{p+1, \ldots, p+k\}\}$ always move synchronously with master agent $A_{1,0}$ and the additional agents $\mathcal{A}'_2 = \{A_{2,i} : i \in \{p+1, \ldots, p+k\}\}$ always move synchronously with master agent $A_{2,0}$. As an effect, the additional agents $\mathcal{A}'_j$ always have the same incoming port number as the corresponding master agent $A_{j,0}$. The states 0 and 1 of the additional agents can be used to store the state of agent A so that a constant number of states of the master agent suffice. When the master agent $A_{j,0}$ reaches a halting state the additional agents in $\mathcal{A}'_j$ also switch to their halting state $\bar{h}$ so that they also terminate when the exploration is finished. □

We note that both the pickup invariant and the doubling of the number of agents are unnecessary in a slightly stronger model of cooperative exploration where the transition function of one agent may also depend on the incoming labels of other agents at the same vertex. In that case, pebble agents may simply remain at the vertex where they were dropped and resume computation when the master agent returns to the vertex. It is further worth noting that our lower bound construction of Section 4 also remains valid for this slightly stronger model.

3 EXPLORATION ALGORITHMS

In this section, we devise an agent exploring any graph on at most n vertices with $O(\log\log n)$ pebbles and $O(\log\log n)$ memory that maintains the pickup invariant and terminates while carrying all pebbles. By the reductions between the agents' models given in Section 2 this implies that (a) an agent with $O(\log\log n)$ pebbles and constant memory can explore any n-vertex graph and (b) that a set of $O(\log\log n)$ agents with constant memory each can explore any n-vertex graph.

For our algorithm, we use the concept of *exploration sequences* (Koucký [28]). An exploration sequence is a sequence of integers $e_0, e_1, e_2, \ldots$ that guides the walk of an agent through a graph G as follows: Assume an agent starts in a vertex v_0 of G and let $l_0 = \perp$. Let v_i denote the agent's location in step i and l_i the port number of the edge at v_i leading back to the previous location. Then, the agent *follows* the exploration sequence $e_0, e_1, e_2, \ldots$ if, in each step i, it traverses the edge with port number $(l_i + e_i) \bmod d_{v_i}$ at v_i to the next vertex v_{i+1}, where d_{v_i} is the degree of v_i (for the first step, port number $e_0 \bmod d_{v_0}$ is chosen instead). An exploration sequence is *universal* for a class of undirected, connected, locally edge-labeled graphs $\mathcal{G}$ if an agent following it explores every graph $G \in \mathcal{G}$ for any starting vertex in G, i.e., for any starting vertex it visits all vertices of G. For a set S, we further use the notation $S^+ := \bigcup_{i=1}^{\infty} S^i$ to denote the set of finite sequences with elements in S. The following fundamental result of Reingold [33] establishes that universal exploration sequences can be constructed in logarithmic space.

Theorem 3.1 ([33], Corollary 5.5). *There exists an algorithm taking n and d as input and producing in $O(\log n)$ space a finite exploration sequence universal for all connected d-regular graphs on n vertices.*

Reingold's result implies in particular that there is an agent without pebbles and $O(n^c)$ states for some constant c that explores any d-regular graph with n vertices when both n and d are known. We further note that Reingold's algorithm can be implemented on a Turing machine that has a read/write tape of length $O(\log n)$ as work tape and writes the exploration sequence to a write-only output tape, see Reference [33, Section 5] for details. For formal reasons the Turing machine in [33] additionally has a read-only input tape from which it reads the values of n and d encoded in unary so that the space complexity of the algorithm is actually logarithmic in the input length. For our setting, it is sufficient to assume that n and d are given as binary encoded numbers on the working tape of length $O(\log n)$, as we care only about the space complexity of exploration in terms of the number of vertices n.

As a first step, we show in Lemma 3.2 how to modify Reingold's algorithm for 3-regular graphs to yield a closed walk containing an exponential number of vertices in terms of the memory used. Afterwards, we extend this result to general graphs in Lemma 3.3.

Lemma 3.2. *For any $z \in \mathbb{N}$, there exists an $O(\log z)$-space algorithm producing an exploration sequence $w \in \{0, 1, 2\}^+$ such that for all connected 3-regular graphs G with n vertices the following hold:*

(1) An agent following w in G explores at least $\min\{z, n\}$ distinct vertices.
(2) An agent that starts in a vertex v_0 with incoming port number l_0 returns to v_0 with incoming port number l_0 after following w. In particular, w yields a closed walk in G.
(3) The length of w is bounded by $z^{O(1)}$.

Proof. By Theorem 3.1, there is a Turing machine M_0 with a tape of length $O(\log z)$ producing a finite universal exploration sequence $e_0, e_1, \ldots, e_{a-1}$ of length $a \in \mathbb{N}$ for any 3-regular graph on exactly $4z$ vertices.

The Turing machine M producing an exploration sequence w with the desired properties is given in Algorithm 1. By construction, the sequence w produced by M is

$$e_0, e_1, \ldots, e_{a-1}, 0, (-e_{a-1} \bmod 3), (-e_{a-2} \bmod 3), \ldots, (-e_1 \bmod 3), (-e_0 \bmod 3), 0.$$

We first show the second property. Let an agent A start at a vertex v_0 with incoming port number l_0 in some 3-regular graph G. Let further A follow the exploration sequence w, and, for $i \in \{0, \ldots, a-1\}$, let v_{i+1} be the vertex reached after following w up to e_i. Then the offset 0 takes the agent back from v_a to v_{a-1} and afterwards $-e_i \bmod 3$ takes agent A from v_i to v_{i-1}. This means that the offset $-e_1 \bmod 3$ takes A back to v_0 with incoming port number $l_0 + e_0$. Hence, after the offsets $-e_0 \bmod 3$ and 0, agent A has returned to v_0 with incoming port number l_0. This yields the second property.

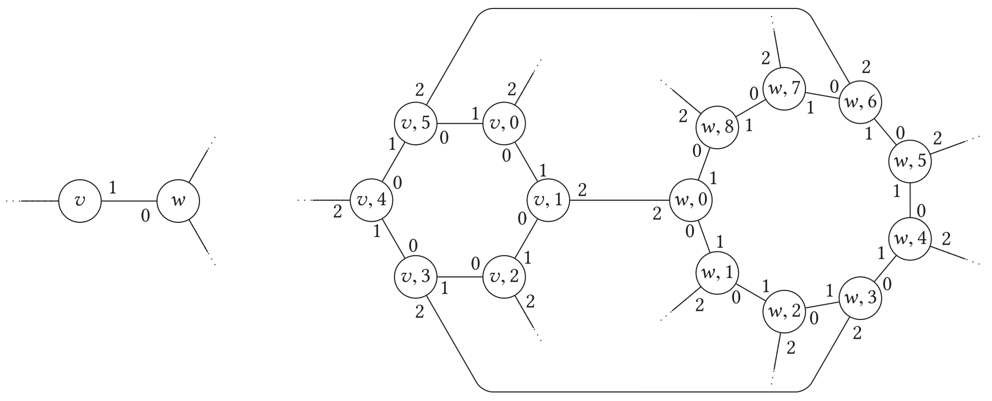
Moreover, the length a of the exploration sequence of M_0 is bounded by the number of configurations of M_0, i.e., the number of possible combinations of state, head position, and tape contents. The working tape has length $O(\log z)$. Therefore, the number of configurations of M_0 and hence also a is bounded by $z^{O(1)}$, which yields the third property. As the auxiliary variable t ranges from 1 to $2a + 2$ and running the Turing machine M_0 for t steps can be implemented in $O(\log z)$ space, the Turing machine M can be implemented to run in $O(\log z)$ space.

It is left to show that an agent following w in an arbitrary connected 3-regular graph with n vertices explores at least $\min\{z, n\}$ vertices. For the sake of contradiction, assume there exists some 3-regular graph G on n vertices so that an agent A starting in a vertex v_0 and following the exploration sequence w produced by M only visits a set of vertices V_0 with $|V_0| < \min\{z, n\}$. Let G_0 be the subgraph of G induced by V_0. Note that, since $|V_0| < n$ by assumption, at least one vertex in G_0 has degree less than 3. We now extend G_0 to a connected 3-regular graph with $4z$ vertices as follows. First, we let G_1 be the graph G_0 after adding an isolated vertex if $|V_0|$ is odd, and we let V_1 be the vertex set of G_1. We further let G_2 be a cycle of length $4z - |V_1|$ with opposite vertices connected by an edge. Note that $4z$ and $|V_1|$ are even and G_2 is 3-regular. As long as G_1 contains at least one vertex of degree less than 3, we delete an edge $\{w, w'\}$ connecting opposite vertices in the cycle in G_2 and for w and then w' add an edge from this vertex to a vertex of degree less than 3 in G_1 (possibly the same). This procedure terminates when all vertices in G_1 have degree 3, since G_2 contains $4z - |V_1| \geq 3z \geq 3|V_1|$ vertices and there cannot be a single vertex of degree 2 left in G_1, as this would mean that the sum of all vertex degrees in G_1 is odd. The labels in $\{0, 1, 2\}$ at both endpoints of every edge not in G_0 are chosen arbitrarily. Let H be the resulting 3-regular graph with $4z$ vertices containing G_0 as induced subgraph.

By construction, the walk of an agent A starting in H at v_0 and following w is the same as the walk in G starting in v_0 and following w. In particular, the agent A only visits the vertices V_0 and does not explore H. This contradicts that the sequence $e_0, e_1, \ldots, e_{a-1}$, which corresponds to the first a elements of the exploration sequence w, is a universal exploration sequence for all connected 3-regular graph on $4z$ vertices by assumption. □

ALGORITHM 1: Turing machine M computing exploration sequence for 3-regular graphs.

```
for $t \in \{1, \ldots, 2a + 2\}$ do
    if $t \leq a$
        run $M_0$ for $t$ steps to obtain element $e_{t-1}$ of the exploration sequence generated by $M_0$
        output $e_{t-1}$
    else if $t = a + 1$ or $t = 2a + 2$
        output 0
    else if $a + 2 \leq t \leq 2a + 1$
        run $M_0$ for $2a + 1 - t$ steps to obtain element $e_{2a+2-t}$ of exploration sequence of $M_0$
        output $-e_{2a+1-t} \bmod 3$
```



(a) Original graph G. (b) 3-regular graph G_{reg}.

Fig. 1. Example for the transformation of a graph G to a 3-regular graph G_{reg}. A vertex v of degree 2 is transformed to a cycle containing six vertices and for the edge $\{v, w\}$, three edges are added to the graph.

We proceed to give a similar result for non-regular graphs.

Lemma 3.3. *For any $z \in \mathbb{N}$, there exists a $O(\log z)$-space algorithm producing an exploration sequence $w \in \{-1, 0, 1\}^+$ such that for all connected graphs G with n vertices the following hold:*

(1) An agent following w in G explores at least $\min\{z, n\}$ distinct vertices.
(2) An agent that starts in a vertex v_0 with incoming port number l_0 returns to v_0 with incoming port number l_0 after following w. In particular, w yields a closed walk in G.
(3) The length of w is bounded by $z^{O(1)}$.

Proof. Let M_{reg} be the Turing machine of Lemma 3.2 with a tape of length bounded by $O(\log z)$ producing a universal exploration sequence $w_{\text{reg}} \in \{0, 1, 2\}^+$ such that an agent following w_{reg} in some 3-regular graph with n vertices visits at least $\min\{3z^2, n\}$ distinct vertices.

To prove the statement, we transform this universal exploration sequence for 3-regular graphs to a universal exploration sequence universal for general graphs by using a construction taken from Koucký [29, Theorem 87]. In this construction, an arbitrary graph G with n vertices is transformed into a 3-regular graph G_{reg} as follows: We replace every vertex v of degree d_v by a circle of $3d_v$ vertices $(v, 0), \ldots, (v, 3d_v - 1)$, where the edge $\{(v, i), (v, i + 1 \bmod 3d_v)\}$ has port number 0 at (v, i) and port number 1 at $(v, i + 1 \bmod 3d_v)$, see also Figure 1 for an example of this construction. For any edge $\{v, w\}$ in G with port number i at v and j at w, we add the three edges $\{(v, i), (w, j)\}, \{(v, i + d_v), (w, j + d_w)\}, \{(v, i + 2d_v), (w, j + 2d_w)\}$ with port numbers 2 at both endpoints to G_{reg}.

Observe that there are only two labelings of edges in G_{reg}, edges with port number 2 at both endpoints and edges with port numbers 0 and 1. In particular, one port number of an edge can be deduced from the other port number. As a consequence, given the initial incoming port number and the edge offsets from the exploration sequence w_{reg} produced by M_{reg}, all outgoing port numbers can be computed without knowing the incoming port number at every vertex. In other words, we can transform the sequence of edge label offsets given by w_{reg} to a traversal sequence, i.e., a sequence of absolute edge labels $l_0, l_1, \ldots$ of G_{reg}.

We proceed to define the Turing machine M producing an exploration sequence $w \in \{-1, 0, 1\}^+$ with the desired properties as shown in Algorithm 2. We assume that the initial incoming port number is 0 and hence $l_0 = w_{\text{reg}}(0)$. First, note that the next outgoing port number l_i in G_{reg} can be computed from the last outgoing port number in G_{reg} and the offset $w_{\text{reg}}(i)$ in constant space

ALGORITHM 2: Turing machine M computing exploration sequence for arbitrary graphs.

```
1:  output 0,0
2:  $i \leftarrow 0$
3:  while $M_{\text{reg}}$ has not terminated do
4:      obtain next offset $w_{\text{reg}}(i)$ from $M_{\text{reg}}$
5:      compute edge label $l_i$ in $G_{\text{reg}}$
6:      if $l_i = 0$
7:          output 1,0
8:      else if $l_i = 1$
9:          output $-1, 0$
10:     else if $l_i = 2$
11:         output 0
12:     $i \leftarrow i + 1$
```

(line 5 of Algorithm 1). Thus, M can be implemented in $O(\log z)$ space. By assumption, the length of the exploration sequence produced by M_{reg} is bounded by $z^{O(1)}$. Hence, also the length of the exploration sequence produced by M is bounded by $z^{O(1)}$.

What is left to show is the first and second property. Let A be an agent following the exploration sequence w produced by M in G and starting at a vertex v_0 with incoming port number a_0. Let further A_{reg} be an agent following w_{reg} in G_{reg} and starting at vertex (v_0, a_0) with incoming port number 0. We first establish the following invariants that hold after every iteration i of the while-loop in Algorithm 2:

(1) If agent A_{reg} is at vertex (v_i, a_i) in G_{reg} after i steps, then after following the exploration sequence output by M up to the end of iteration i agent A is at v_i in G and $a_i \bmod d_{v_i}$ is the current incoming port number.
(2) If (v_i, a_i) is visited by A_{reg} in G_{reg}, then in G both v_i and the neighbor incident to the edge with label $(a_i \bmod d_{v_i})$ are visited by A.

We show the invariants by induction. Note that at the beginning the Turing machine M outputs 0,0 so that in G agent A visits the neighbor of v_0 incident to the edge with port number a_0 and then returns to v_0. Thus, both invariants hold before the first iteration of the while-loop.

Now assume that before iteration i both invariants hold. We show that then they also hold after iteration i. If agent A_{reg} is at the vertex (v, a) and the edge traversed by A_{reg} in step i has label 0, i.e., $l_i = 0$, then A_{reg} moves to vertex $(v, (a + 1) \bmod 3d_v)$ by the definition of G_{reg}, see also Figure 1. By assumption, agent A is at vertex v in G and the incoming port number is $a \bmod d_v$. Thus, if agent A follows the exploration sequence 1,0 output by M in iteration i (line 7 of Algorithm 2), then it first traverses the edge labeled $(a + 1) \bmod d_v$ and then returns to v. This means that after iteration i, the current vertex of A in G is v and the incoming port number is $(a + 1) \bmod d_v = ((a + 1) \bmod 3d_v) \bmod d_v$. Moreover, agent A visited both v and the neighbor of v incident to the edge with label $(a + 1) \bmod d_v$. Thus, both invariants hold after iteration i in this case.

The case that $l_i = 1$ is analogous except that edges with label $l_i = 1$ in G_{reg} lead from a vertex (v, a) to a vertex $(v, (a - 1) \bmod 3d_v)$. The equivalent movement of A in G is achieved by the sequence $-1, 0$ (line 9 in Algorithm 1).

So assume that agent A_{reg} in step i traverses an edge with label $l_i = 2$ from a vertex (v, a) to a vertex (v', a'). This means that there is an edge $\{v, v'\}$ in G with port number $a \bmod d_v$ at v and port number $a' \bmod d_{v'}$ at v'. By assumption, at the beginning of iteration i agent A is at v and $a \bmod d_v$ is the label of the edge to the previous vertex. So if A follows the exploration sequence 0 output in iteration i (line 11 of Algorithm 2), then it moves to v'. Now the label to the previous vertex at v' is $a' \bmod d_{v'}$ and A visited both v and v' so that both invariants hold again.

Finally, for the second property in the lemma, we know that after following the exploration sequence w_{reg} agent A_{reg} returns to (v_0, a_0) in G_{reg} by Lemma 3.2. Thus, after following w agent A returns to v_0 and a_0 is the incoming port number by the first invariant.

What is left to show is that A visits at least $\min\{z, n\}$ distinct vertices in G. If G_{reg} has at most $3z^2$ vertices, then A_{reg} visits all vertices in G_{reg} by assumption and thus A also visits all vertices in G by the second invariant. Otherwise, we know that A_{reg} visits at least $3z^2$ distinct vertices in G_{reg}. Note that this implies $z < n$ as G_{reg} contains at most $3n(n-1)$ vertices.

Assume, for the sake of contradiction, that A visits less than z vertices in G. Let $\bar{V}_{\text{reg}}$ be the set of vertices visited by A_{reg} in G_{reg}. As $|\bar{V}_{\text{reg}}| \geq 3z^2$ by assumption, at least one of the two following cases occurs:

(1) The cardinality of $\bar{V} := \{ v \mid (v, j) \in \bar{V}_{\text{reg}} \text{ for some } j \}$ is at least z.
(2) There is a vertex $\bar{v}$ in G such that $M_{\bar{v}} := \{ j \mid (\bar{v}, j) \in \bar{V}_{\text{reg}} \}$ has cardinality $\geq 3z$.

We show that both cases lead to a contradiction.

Note that by the second invariant agent A visits all vertices in $\bar{V}$. Thus, if $|\bar{V}| \geq z$, then A visits at least z distinct vertices in G, a contradiction.

Assume the second case occurs and let $\bar{v}$ in G be a vertex such that $|M_{\bar{v}}| \geq 3z$. Then, we have $|\{j \bmod d_{\bar{v}} \mid j \in M_{\bar{v}}| \geq z$ implying that agent A visits at least z neighbors of $\bar{v}$ in G by the second invariant. This again is a contradiction. □

To make the results above usable for our agents with pebbles, we need more structure regarding the memory usage of the agent. To this end, we formally define a walking Turing machine with access to pebbles, which we will refer to as a *pebble machine*. Formally, we can view such a walking Turing machine as a weaker agent model than the general agent model with pebbles described in Section 2.2.2, where the states of the agent correspond to the state of the working tape, the position of the head, and the state of the Turing machine. Specifically, this model is weaker, since it separates computations on its tape from state transitions that depend on pebble locations and incoming port number, and since it demands δ_{TM} to be computable.

Definition 3.4. Let $s, p, m \in \mathbb{N}$. An (s, p, m)*-pebble machine* $T = (Q, \bar{Q}, P, m, \delta_{\text{in}}, \delta_{\text{TM}}, \delta_{\text{out}}, q^*)$ is an agent $A = (\Sigma, \bar{\Sigma}, \delta, \sigma^*)$ with a set $P = \{1, \ldots, p\}$ of p pebbles and the following properties:

(1) The set of states is $\Sigma = Q \times \{0,1\}^m \times \{0, \ldots, m-1\}$, where $|Q| = s$. This means that each agent state consists of a Turing state, the state of the working tape of length m, and a head position on the tape.
(2) In the initial state σ^* the Turing state is q^*, the head position is 0, and the tape has 0 at every position.
(3) The agent's transition function $\delta : \Sigma \times \mathbb{N} \times (\mathbb{N} \cup \{\perp\}) \times 2^P \times 2^P \rightarrow \Sigma \times (\mathbb{N} \cup \{\perp\}) \times 2^P \times 2^P$ is computed as follows:
    (a) The agent first observes its local environment according to the function $\delta_{\text{in}} : Q \times \mathbb{N} \times (\mathbb{N} \cup \{\perp\}) \times 2^P \times 2^P \rightarrow Q$ that maps a vector (q, d_v, l, P_A, P_v) consisting of the current Turing state, the degree d_v of the current vertex, the label l of the edge leading back to the vertex last visited, the set P_A of carried pebbles and the set P_v of pebbles located at the current vertex to a new Turing state q'.
    (b) The agent does computations on the working tape like a regular Turing machine according to the function $\delta_{\text{TM}} : Q \times \{0,1\} \rightarrow Q \times \{0,1\} \times \{\text{left}, \text{right}\}$ that maps the tuple consisting of the current Turing state and the symbol at the current head position (q, a) to a tuple (q', a', d) meaning that the machine transitions to the new state q',

writes a' at the current position of the head and moves the head in direction d; this process is repeated until a halting state $\bar{q} \in \bar{Q}$ is reached (note that we only consider Turing machines that eventually halt).

(c) It performs actions according to the function $\delta_{\text{out}} : \bar{Q} \times \mathbb{N} \times (\mathbb{N} \cup \{\perp\}) \times 2^P \times 2^P \to 2^P \times 2^P \times (\mathbb{N} \cup \{\perp\})$ that maps a tuple (q, d_v, l, P_A, P_v) containing the current Turing state q, the degree d_v of the current vertex, the label l of the edge leading back to the vertex last visited, the set of carried pebbles P_A and the set of pebbles P_v at the current vertex v to a tuple (P'_A, P'_v, l'). This means that the agent drops and retrieves pebbles such that it carries P'_A, leaves P'_v at v and traverses the edge with local edge label l'.

When considering a pebble machine $T = (Q, \bar{Q}, P, m, \delta_{\text{in}}, \delta_{\text{TM}}, \delta_{\text{out}}, q^*)$, we call the Turing states Q simply *states*, and we call the set of states Σ of the underlying agent model *configurations*. As the configuration of a pebble machine is fully described by the (Turing) state $q \in Q$, the head position, and the state of the working tape, it has $sm2^m$ configurations. We further call a transition of the agent according to the transition function δ_{TM} a *computation step.*

Note that an agent remains at the same vertex and only changes its configuration when performing a computation step.

We assume that a pebble machine does not forget the incoming port number when remaining at a vertex, i.e., the incoming port number does not become $\perp$ in this case. Note that for any pebble machine T there is an agent A with pebbles that never waits at a vertex and combines multiple intermediate transitions of the pebble machine (of which only the last one results in an edge traversal) into one transition with edge traversal. Then the agent A always has access to the incoming port number, so we may make this assumption for the sake of simplicity and without strengthening the agent model.

In the following theorem, we explain how to place pebbles on a closed walk and use them as additional memory.

Theorem 3.5. *There are constants $c, c' \in \mathbb{N}$ such that the following holds: Let T be a $(s, p, 2m)$-pebble machine that performs a closed walk in any graph following an exploration sequence in $\{-1, 0, 1\}^+$ and terminates with all p pebbles at the starting vertex. Then there exists a $(cs, p + c, m)$-pebble machine T' that follows an exploration sequence in $\{-1, 0, 1\}^+$ and terminates at the starting vertex carrying all $p + c$ pebbles. Moreover, the following properties hold:*

(1) For every graph G with $n < 2^{m/c'}$ vertices, the pebble machine T' explores G. The overall number of edge traversals and computation steps needed by the pebble machine T' is bounded by $2^{O(m)}$.

(2) For every graph G with $n \geq 2^{m/c'}$ vertices, T' reproduces the walk of T in G. For the initialization, T' needs $2^{O(m)}$ edge traversals and computations steps. Afterwards, the number of edge traversals and computation steps needed by the pebble machine T' to reproduce one edge traversal or computation step of T is bounded by $2^{O(m)}$.

(3) If T satisfies the pickup invariant, then so does T'.

Proof. The general idea of the proof is that T' places the constant number of additional pebbles on a closed walk ω to encode the tape content of the pebble machine T. Using these pebbles, T' can also count the number of distinct vertices on the closed walk ω. If the closed walk is too short, then T' already explored the graph and the first case occurs. Otherwise, the closed walk is long enough to allow for a sufficient number of distinct positions of the pebble, and we are in the second case of the statement of the theorem.

Let Q be the set of states of T. We define the set of states of T' to be $Q \times Q'$ for a set Q', i.e., every state of T' is a tuple (q, q'), where q corresponds to the state of T in the current step of the traversal. The pebble machine T' observes the input according to δ_{in}, performs actions according to δ_{out}, and uses p pebbles in the same way as T. Thus, after the transitions that correspond to the transitions of pebble machine T, the first component of the state of T' corresponds to the state of T. Moreover, the positions of the p pebbles that T' and T have in common is the same. Additionally, the pebble machine T' makes many intermediate transitions to simulate the computations of T and to carry the additional pebbles $\{p_0, p_1, \dots, p_{c-2}, p_{\text{temp}}\}$ along. For reproducing a step of the pebble machine T, the pebbles $\{p_0, p_1, \dots, p_{c-2}\}$ are placed along a closed walk ω to simulate the memory of T, while the states Q' and the tape of T' are used to manage this memory. The purpose of the pebble p_{temp} will be explained later.

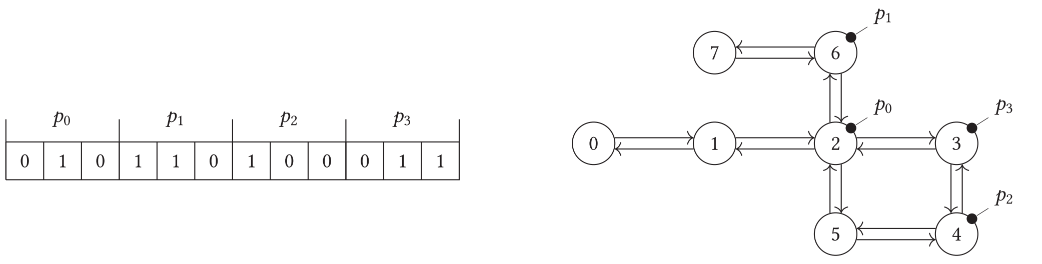
We divide the tape of T' into a constant number c_0 of blocks of size m/c_0 each. In the course of the proof, we will introduce a constant number of variables to manage the simulation of the memory of T with pebbles. Each of these variables is stored in a constant number of blocks. The constant c_0 is chosen large enough to accommodate all variables on the tape of T'. By Lemma 3.3, there is a constant c_1 such that for any $r \in \mathbb{N}$ there is a Turing machine M with at most c_1 states and a tape of length $c_1 \cdot r$ outputting an exploration sequence that gives a closed walk of length at most $2^{c_1 \cdot r}$ visiting at least $\min\{2^r, n\}$ vertices in any graph with n vertices. Let $m_1 := m/(c_0 c_1)$ and let $m_0 \in \mathbb{N}$ be such that for all $m' \in \mathbb{N}$ with $m' \geq m_0$, we have $c_1 \leq 2^{m'/c_0}$ and $2^{m'/c_0} > 2m'$.

In the following, we show how the simulated memory is managed by providing algorithms in pseudocode (see Algorithms 3 to 6). These can be implemented on a Turing machine with a constant number of states c_{Alg}. Let $c = \max\{2^{2m_0}, 2c_0c_1 + 1, c_{\text{Alg}}\}$ and $c' := c_0c_1$. Note that c only depends on the constants c_0, c_1 and c_{Alg}, but not on m or p. It is without loss of generality to assume $m \geq m_0$, because for $m < m_0$, we can store the configuration of the tape of T in the states Q' of T', since $c \geq 2^{2m_0}$.

We proceed to show that the computations on the tape of length $2m$ performed by T according to the transition function δ_{TM} can be simulated using the pebbles $\{p_0, p_1, \dots, p_{c-2}, p_{\text{temp}}\}$. The proof of this result proceeds along the following key claims.

(1) We can find a closed walk ω that starts at the current vertex v and contains 2^{m_1} distinct vertices so that $c - 1$ pebbles placed along this walk can encode all configurations of the tape of T.
(2) We can move along ω while keeping track of the number of steps and counting the number of distinct vertices until we have seen 2^{m_1} distinct vertices.
(3) We can read from and write to the memory encoded by the placement of the pebbles along ω.
(4) If T moves from vertex v to vertex v', then we can move all pebbles to a closed walk ω' starting in v' while preserving the content of the memory.

1. Finding a closed walk ω. Lemma 3.3 yields a Turing machine M_{walk} with c_1 states and a tape of length m/c_0 that produces an exploration sequence corresponding to a closed walk ω that contains at least $\min\{n, 2^{m_1}\}$ distinct vertices and has length at most $2^{c_1 m_1} = 2^{m/c_0}$. We use a variable R_{walk} of size m/c_0 for the memory of M_{walk}, which is initially assumed to have all bits set to 0. If $2^{m_1} > n$, then the exploration sequence produced by M_{walk} is a walk exploring G. Note that by definition $m/c' = m_1$ so this happens exactly when the first case in the theorem occurs. Below, we will show how to count the number of unique vertices on the closed walk of M_{walk}. Hence, the pebble machine T' can initially walk along the closed walk ω counting the number of distinct vertices. If this number is smaller than 2^{m_1}, then we know that we have visited all vertices of G. In this case, all pebbles used can be picked up while once walking along the closed walk ω. We show at the end of the proof that this takes at most $2^{O(m)}$ edge traversals and computation steps.



(a) Tape memory (b) Memory encoded by pebbles

Fig. 2. Memory encoding by pebbles on a closed walk. The state of the tape of length $2m = 12$ in (a) is encoded by the position of the $c - 1 = 4$ pebbles in (b). The number of the vertices corresponds to the order of first traversal by the closed walk ω starting in 0. The position of each pebble encodes $m_1 = 3$ bits.

ALGORITHM 3: Auxiliary functions for moving along the closed walk ω.

```
function Step()
    traverse edge according to value of exploration sequence output by $M_{\text{walk}}$
    $R_{\text{steps}} \leftarrow R_{\text{steps}} + 1$

function FindPebble($p_i$)
    while not observe($p_i$) do
        Step()

function Restart()
    while $M_{\text{walk}}$ has not terminated do
        Step()
    $R_{\text{steps}} \leftarrow 0$
    $R_{\text{id}} \leftarrow 0$
    $R_{\text{walk}} \leftarrow 0$
```

From now on, we can therefore assume that ω contains at least 2^{m_1} distinct vertices. We need to show that $c - 1$ pebbles placed along the walk ω can be used to encode all of the 2^{2m} configurations of the tape of T. Figure 2 shows how each pebble encodes a certain part of the tape of T. The idea is that each pebble can be placed on one of 2^{m_1} different vertices, thus encoding exactly m_1 bits. We divide the tape of length $2m$ into $2m/m_1 = 2c_0c_1$ parts of size m_1 each, such that the position of pebble p_i encodes the bits $\{im_1, \ldots, (i+1)m_1 - 1\}$, where we assume the bits of the tape of T to be numbered $0, 1 \ldots, 2m - 1$. As $c \geq 2c_0c_1 + 1$, we have enough pebbles to encode the configuration of the tape of T.

2. Navigating ω. Let R_{steps} be a variable counting the number of steps along ω and R_{id} be a variable for counting the number of unique vertices visited along ω starting in v. Note that R_{id} gives a way of associating a unique identifier to the first 2^{m_1} distinct vertices along ω. As $m_1 \leq m/c_0$ holds, m/c_0 tape cells suffice for counting the first 2^{m_1} distinct vertices along ω. The overall number of steps along the closed walk is bounded by $2^{m/c_0}$ and therefore m/c_0 tape cells also suffice for counting the steps along ω.

It remains to show that we can move along the closed walk ω while updating R_{steps} and R_{id}, such that, starting from the vertex v, the variable R_{steps} contains the number of steps taken and R_{id} contains the number of distinct vertices visited. Let drop(p_i) denote the operation of dropping pebble p_i at the current location, pickup(p_i) the operation of picking up p_i at the current location

ALGORITHM 4: Moving along the closed walk ω while updating R_{steps} and R_{id}.

```
function NextDistinctVertex()
    if $R_{\text{id}} = 2^{m_1} - 1$
        Restart()
        return
    $R_{\text{id}} \leftarrow R_{\text{id}} + 1$
    while True do
        Step()
        drop($p_{\text{temp}}$)
        $R'_{\text{steps}} \leftarrow R_{\text{steps}}$
        Restart()
        FindPebble($p_{\text{temp}}$)
        if $R_{\text{steps}} = R'_{\text{steps}}$
            pickup($p_{\text{temp}}$)
            break
        while $R_{\text{steps}} < R'_{\text{steps}}$ do
            Step()
        pickup($p_{\text{temp}}$)
```

if possible, and let OBSERVE(p_i) be "true" if and only if pebble p_i is located at the current position. Consider the auxiliary functions shown in Algorithms 3. The function Step() moves one step along ω and updates R_{steps} accordingly. The function FindPebble(p_i) moves along ω until it finds pebble p_i. The function Restart() follows the exploration sequence output by M_{walk} until M_{walk} terminates. After M_{walk} terminates, the pebble machine T has returned to the starting vertex v and the incoming edge label is again the same as at the beginning of the closed walk ω by Lemma 3.3. The variables R_{steps} and R_{id} are then set to 0, and M_{walk} is restarted by setting the variable R_{walk} to 0. Finally, the function NextDistinctVertex() in Algorithm 4 does the following: If the number of distinct vertices visited along ω is already 2^{m_1}, then the pebble machine T' returns to the start. Otherwise, it continues along ω until it encounters a vertex that it has not visited before. It repeatedly traverses an edge, drops the pebble p_{temp}, stores the number of steps until reaching that vertex, then restarts from the beginning and checks if it can reach the vertex containing pebble p_{temp} with fewer steps. If not, then the while-loop can be exited as T' found a new distinct vertex. Note that we use the auxiliary variable R'_{steps}, which needs a constant number of blocks of size m_0/c_0.

3. Reading from and writing to simulated memory. We show how to simulate the changes to the tape of T by changing the positions of the pebbles along ω. The transition function δ_{TM} of T determines how T does computations on its tape and, in particular, how T changes its head position. We use a variable R_{head} of size m/c_0 to store the head position. By assumption, $m \geq m_0$ and therefore $2^{m/c_0} > 2m$, i.e., the size of R_{head} is sufficient to store the head position. To simulate one transition of T according to δ_{TM}, we need to read the bit at the current head position and then write to the simulated memory and change the head position accordingly. Reading from the simulated memory is done by the function ReadBit() and writing of a bit b to the simulated memory by the function WriteBit(b) (cf. Algorithms 6).

First, let us consider the two auxiliary functions GetPebbleId(p_i) and PutPebbleAtId(p_i, id) (cf. Algorithms 5). As the name suggests, the function GetPebbleId(p_i) returns the unique identifier associated to the vertex marked by p_i. Recall that vertices are indistinguishable. Here, unique identifier refers to the number of distinct vertices on the walk ω before reaching the vertex marked with p_i for the first time. Given an identifier id, we can use the function PutPebbleAtId(p_i, id)

ALGORITHM 5: Reading and changing positions of pebbles.

```
function GetPebbleId($p_i$)
    Restart()
    while not observe($p_i$) do
        NextDistinctVertex()
    return $R_{\text{id}}$

function PutPebbleAtId($p_i$,id)
    Restart()
    FindPebble($p_i$)
    pickup($p_i$)
    Restart()
    while id>0 do
        id ← id − 1
        NextDistinctVertex()
    drop($p_i$)
```

ALGORITHM 6: Reading and writing one bit for the simulated memory.

```
function ReadBit()
    $i \leftarrow \lfloor R_{\text{head}}/m_1 \rfloor$
    $j \leftarrow R_{\text{head}} - m_1 \cdot i$
    id ← GetPebbleId($p_i$)
    return $j$-th bit of id (in binary)

function WriteBit($b$)
    $i \leftarrow \lfloor R_{\text{head}}/m_1 \rfloor$
    $j \leftarrow R_{\text{head}} - m_1 \cdot i$
    id ← GetPebbleId($p_i$)
    if $b = 1$ and ReadBit() = 0
        id ← id + $2^j$
    else if $b = 0$ and ReadBit() = 1
        id ← id − $2^j$
    PutPebbleAtId($p_i$,id)
```

for placing pebble p_i at the unique vertex corresponding to id. By the choice of our encoding, if $R_{\text{head}} = i \cdot m_1 + j$ with $j \in \{0, \ldots, m_1 - 1\}$, then the j-bit of the binary encoding of the position of pebble p_i encodes the contents of the tape cell specified by R_{head}. Thus, for reading from the simulated memory, we have to compute i and j and determine the position of the corresponding pebble in the function ReadBit(). For the function WriteBit(b), we also compute i and j. Then, we move the pebble p_i by 2^j unique vertices forward if the bit flips to 1 or by 2^j unique vertices backward if the bit flips to 0.

4. *Relocating ω.* Assume T is at vertex v with current incoming edge label l and it moves to another vertex v'. By assumption, T follows an exploration sequence in $\{-1, 0, 1\}^+$ such that there is an offset $l_0 \in \{-1, 0, 1\}$ and T traverses the edge with port number $l_1 := l + l_0 \bmod d_v$ at v. We further let l' be the incoming port number at v'. Due to Lemma 3.3, after every traversal of the closed walk ω, pebble machine T' returns to v with incoming edge label l. After having computed the label l_1 of the edge to v', T' can move between vertex v with incoming edge label l and vertex v' with incoming edge label l' using constant memory and without the need to recompute l_1: The offset $l_0 \in \{-1, 0, 1\}$ takes T' from vertex v and incoming edge label l to vertex v' with incoming

edge label l'. Moreover, the sequence of offsets $0, -l_0, 0$ takes T' from vertex v' and incoming edge label l' back to vertex v with incoming edge label l.

Hence, T' can move the pebbles placed along the walk ω to the corresponding positions along a new walk ω' starting at v' with incoming edge label l' in the following way: We iterate over all $c - 1$ pebbles and for each pebble p_i, we start in v, determine the identifier id of the vertex marked by p_i via GetPebbleId(p_i), pick up p_i, move to v' and place p_i on ω' using the function PutPebbleAtId(p_i, id). This way, we can carry the memory simulated by the pebbles along during the graph traversal. Note that as soon as pebble machine T terminates at a vertex v, the pebble machine T' can simply move once along the closed walk ω, pick up all pebbles, return to v and terminate. By assumption, all p pebbles of T are at v when T terminates and by construction also the additional c pebbles are at v when T' terminates.

Thus, we have shown that in the second case T' can reproduce the traversal of T in G while using a tape with half the length, but c additional pebbles and a factor of c additional states.

We now bound the number of edge traversals and computation steps in both cases. First, we bound the number of edge traversals that T' needs for simulating one computation step of T. Recall that T' needs at most $2^{m/c_0} \leq 2^m$ edge traversals for moving once along the whole closed walk ω. A call of the function Step() corresponds to one edge traversal, a call of FindPebble(p_i) thus corresponds to at most 2^m edge traversals and also a call of Restart() corresponds to at most 2^m edge traversals. Moreover, one iteration of the loop in NextDistinctVertex() accounts for at most $2^m + 1$ edge traversals and therefore executing the whole function results in at most $(2^m + 1) \cdot 2^m = 2^{O(m)}$ edge traversals. This means that one call of GetPebbleId(p_i) or PutPebbleAtId(p_i,id) incur at most $2^{O(m)}$ edge traversals and this also holds for ReadBit() and WriteBit(b). Hence, for every computation step performed by T according to δ_{TM}, the pebble machine T' performs actions according to ReadBit() and WriteBit(b) and overall does at most $2^{O(m)}$ edge traversals. The above argument also shows that at most $2^{O(m)}$ edge traversals are necessary to count the number of distinct vertices on the closed walk ω at the beginning.

Next, let us bound the number of edge traversals that T' needs for reproducing one edge traversal of T. This means that we need to count how many edge traversals are necessary to relocate all pebbles placed along the walk ω to the new walk ω'. For every pebble p_i, we call GetPebbleId(p_i), which results in at most $2^{O(m)}$ edge traversals, we pick up p_i and move to v', which again needs at most $2^{O(m)}$ edge traversals, and place p_i on ω' using the function PutPebbleAtId(p_i, id), which also needs $2^{O(m)}$ edge traversals. Overall, this procedure is done for a constant number of pebbles and hence requires at most $2^{O(m)}$ edge traversals.

We now bound the number of computation steps of T' by using the bounds on the number of edge traversals. Recall that the state of T' is a tuple (q, q'), where q corresponds to the state of T. In the computation only the second component of the state of T' changes and therefore there are only at most c possible states. The tape length and number of possible head positions of the Turing machine is m. Since we may assume without loss of generality that $m \geq 2$, we can bound the number of distinct configurations of T' in each computation by $2^{O(m)}$. Hence, after every edge traversal T' does at most $2^{O(m)}$ computation steps. This implies that in the first case of the statement of the theorem, the number of computation steps is bounded by $2^{O(m)}$, because the number of edge traversals is bounded by $2^{O(m)}$ as shown above. Similarly, in the second case of the statement of the theorem the total number of computation steps after $2^{O(m)}$ edge traversals is bounded by $2^{O(m)}$. Since $m \geq 2$ this means that also the sum of computation steps and edge traversals can be bounded by $2^{O(m)}$ both for one computation step and one edge traversal of T.

Finally, we need to show that the pickup invariant is maintained. By assumption, T satisfies this property. As pebble machine T' drops and picks up the p common pebbles at the same vertices with the same incoming labels, we need to only show this property for the c additional pebbles. For any vertex u on the closed walk ω, let l_u be the incoming edge label when T' first visits u on the closed walk. If u is the current vertex after a call of the function NextDistinctVertex() and R_{steps} is the current number of steps from the starting vertex on the closed walk ω, then we know that u cannot be reached with less than R_{steps} steps. In particular, the incoming edge label must be l_u. Hence, every time T' drops a pebble p_i for $i \in \{0, 1, \ldots c-2\}$ at vertex u, the incoming edge label is l_u and the same holds every time a pebble p_i is picked up from u. Furthermore, the function NextDistinctVertex() ensures that if pebble p_{temp} is dropped after R'_{steps} steps on the closed walk, then it is also picked up again after R'_{steps} steps on the closed walk. Thus, the pickup invariant also holds for the pebble p_{temp}. □

Finally, we show that by recursively simulating a pebble machine by another pebble machine with half the memory but a constant number of additional pebbles we can explore any graph with at most n vertices while using $O(\log \log n)$ pebbles and only $O(\log \log n)$ bits of memory.

Theorem 3.6. *Any connected undirected graph on at most n vertices can be explored by an agent in a polynomial number of steps using $O(\log \log n)$ pebbles and $O(\log \log n)$ bits of memory. The agent does not require n as input and terminates at the starting vertex with all pebbles after exploring the graph. The agent further maintains the pickup invariant.*

Proof. Let $c, c' \in \mathbb{N}$ be the constants of Theorem 3.5. Let $r \in \mathbb{N}$ be arbitrary and consider a $(c, 0, c'2^{r+1})$-pebble machine $T^{(r)}$ that simply terminates without making a computation step or edge traversal. Applying Theorem 3.5 for the pebble machine $T^{(r)}$ gives a $(c^2, c, c'2^r)$-pebble machine $T_r^{(r)}$ that follows an exploration sequence in $\{-1, 0, 1\}^+$ and terminates with all pebbles at the starting vertex. Moreover, if $n < 2^{2^r}$ holds, then $T_r^{(r)}$ explores the graph and returns to the starting vertex. If, however, $n \geq 2^{2^r}$, then $T_r^{(r)}$ reproduces the walk of $T^{(r)}$ (which in this case is of course trivial). Note that these properties hold even though the number n of vertices is unknown and, in particular, not given as input to $T_r^{(r)}$.

Applying Theorem 3.5 iteratively, we obtain a $(c^{r+2-i}, (r+1-i)c, c'2^i)$-pebble machine $T_i^{(r)}$ that follows an exploration sequence in $\{-1, 0, 1\}^+$ and terminates with all pebbles at the starting vertex. For a graph G with $n < 2^{2^r}$, $T_r^{(r)}$ explores G. Thus, for such a graph G it does not matter which case occurs when applying Theorem 3.5, as in both cases, we can conclude that $T_i^{(r)}$ for $i \in \{0, \ldots, r-1\}$ explores the graph G. If we have $n \geq 2^{2^r}$, then $n \geq 2^{2^i}$ holds for all $i \in \{0, \ldots, r-1\}$ and in particular $T_0^{(r)}$ reproduces the walk of $T^{(r)}$ in G.

The desired pebble machine T exploring any graph G with $O(\log \log n)$ pebbles and $O(\log \log n)$ bits of memory works as follows: We have a counter r, which is initially 1 and is increased by one after each iteration until the given graph G is explored. In iteration r, pebble machine T does the same as the $(c^{r+2}, (r+1)c, c')$-pebble machine $T_0^{(r)}$ until it terminates. The pebble machine T terminates as soon as for some $r \in \mathbb{N}$ the pebble machine $T_0^{(r)}$ recognizes that it explored the whole graph. This happens when $r = \lceil \log \log n \rceil + 1$. Hence, T uses at most $O(\log \log n)$ pebbles.

Concerning the memory requirement of T, note that T needs to store the state of $T_0^{(r)}$, the tape content of $T_0^{(r)}$ and the current value of r. There are c^{r+2} states of the pebble machine $T_0^{(r)}$, its tape length is c' and $r \leq \lceil \log \log n \rceil + 1$ in every iteration, so that T can be implemented with $O(\log \log n)$ bits of memory.

It is left to show is that the number of edge traversals of T in the exploration of a given graph G with n vertices is polynomial in n. To this end, we first show that the number of edge traversals of the pebble machine $T_0^{(r)}$ is bounded by $n^{O(1)}$ for all $r \in \{1, \ldots, \lceil \log \log n \rceil + 1\}$. Let $r \in \{1, \ldots, \lceil \log \log n \rceil + 1\}$ be arbitrary and let t_i denote the sum of the number of edge traversals and computation steps of $T_i^{(r)}$ in the given graph G. The pebble machine $T_r^{(r)}$ has a tape of length

$m = c'2^r$. Applying Theorem 3.5, we get that either $T_r^{(r)}$ explores G and uses at most $2^{O(m)}$ edge traversals and computation steps or $T_r^{(r)}$ simulates the walk of a pebble machine that does not make a single edge transition and uses at most $2^{O(m)}$ edges traversals and computation steps. In both cases, we obtain

$$t_r \le 2^{O(2^r)} \le 2^{O(2^{\log\log n})} = 2^{O(\log n)} = n^{O(1)}.$$

This shows the desired bound for t_r. Furthermore, one computation step or one edge traversal of $T_i^{(r)}$ leads to at most $2^{O(c' \cdot 2^i)} = 2^{O(1)2^i}$ edge traversals and computation steps of $T_{i-1}^{(r)}$ by Theorem 3.5. Hence, we obtain

$$t_{i-1} \le 2^{O(1)2^i} t_i \qquad \forall\, i \in \{1, \dots, \lceil \log\log n \rceil + 1\}. \tag{3}$$

By iterative application of Inequality Equation (3), we obtain

$$t_0 \le 2^{O(1)2^i} t_1 \le \cdots \le 2^{O(1)\sum_{i=1}^{\lceil \log\log n\rceil+1} 2^i} \cdot t_{\lceil \log\log n\rceil+1} \le 2^{O(1)2^{\lceil \log\log n\rceil}} \cdot n^{O(1)} \le n^{O(1)}.$$

Thus, the number of edge traversals t_0 of $T_0^{(r)}$ is polynomial in n. As T performs at most $n^{O(1)}$ edge traversals according to $T_0^{(r)}$ for at most $\lceil \log\log n\rceil + 1$ distinct values of r, the overall number of edge traversals of T is also bounded by $n^{O(1)}$.

The pebble machine $T^{(r)}$ satisfies the pickup invariant and, hence, by Theorem 3.5 also the pebble machine $T_0^{(r)}$ satisfies the pickup invariant. For every value of r, the pebble machine $T_0^{(r)}$ returns to the starting vertex carrying all $(r+1)c$ pebbles. Therefore, the constructed pebble machine T picks up all pebbles in the same level of recursion as it drops them and, thus, also satisfies the pickup invariant. ◻

Since an additional pebble is more powerful than a bit of memory (Lemma 2.1), we obtain the following direct corollary of Theorem 3.6.

Corollary 3.7. *Any connected undirected graph on at most n vertices can be explored by an agent in a polynomial number of steps using $O(\log\log n)$ pebbles and constant memory. The agent does not require n as input and terminates at the starting vertex with all pebbles after exploring the graph.*

As the pickup invariant is satisfied by the agent in Theorem 3.6, we can apply Lemma 2.3 and obtain the following corollary.

Corollary 3.8. *Any connected undirected graph on at most n vertices can be explored in polynomial time by a set of $O(\log\log n)$ agents with constant memory each. The agents do not require n as input and terminate at the starting vertex after exploring the graph.*

Remark 1. The agent in Theorem 3.6 requires $O(\log\log n)$ bits of memory and the agents in Corollaries 3.7 and 3.8 only $O(1)$ bits of memory. An interesting question is how much memory is necessary to fully encode the transition function,

$$\delta : \Sigma \times \mathbb{N} \times (\mathbb{N} \cup \{\perp\}) \times 2^P \times 2^P \to \Sigma \times (\mathbb{N} \cup \{\perp\}) \times 2^P \times 2^P,$$

of an agent (see Section 2.2.2). Naively encoding it as a table with a row for every possible state, vertex degree, previous edge label and possible combination of $O(\log\log n)$ pebbles/agents at the current vertex takes $n^{O(1)}$ bits of memory.

However, we can obtain a much more compact encoding by exploiting the specific structure of our algorithm: First, we never explicitly use the degree of the current vertex. Moreover, the Turing machine from Lemma 3.3 that we internally use produces an exploration sequence of the form $\{-1, 0, 1\}^+$. This means that our transition function can be expressed more concisely if we would allow in our model to specify transitions relative to the label of the previous edge.

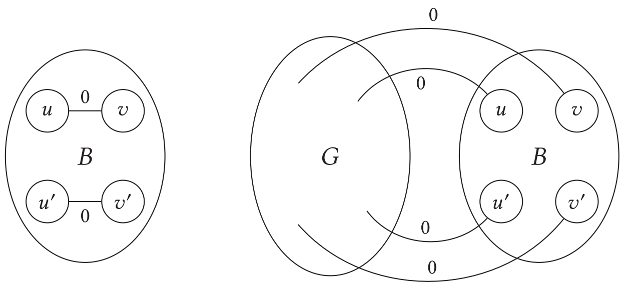


Fig. 3. The r-barrier B on the left with two distinguished edges $\{u, v\}$, $\{u', v'\}$ can be connected to an arbitrary graph G, as shown on the right.

Furthermore, our algorithm only interacts with a constant number of pebbles in every level of the recursion (cf. Theorem 3.5). We can express the state of T in the proof of Theorem 3.6 as a vector, where each component encodes the state in a different level of the recursion. In every transition, only two consecutive entries of this vector can change, as one level of recursion only interacts with the level of recursion below to access the simulated memory.

Since there are only a constant number of states per recursive level, and only a constant number of pebbles involved, all transitions regarding two consecutive levels can be encoded in constant memory. If we therefore explicitly encode all $O(\log \log n)$ levels of recursion and additionally allow to only give the edge label offset in the transition function, then the entire transition function can be encoded with $O(\log \log n)$ bits of memory.

4 LOWER BOUND FOR COLLABORATING AGENTS

The goal in this section is to obtain a lower bound on the number k of s-state agents needed for exploring any graph on at most n vertices. To this end, we will construct a *trap* for a given set of agents, i.e., a graph that the agents are unable to explore. The number of vertices of this trap yields a lower bound on the number of agents required for exploration. The graphs involved in our construction are 3-regular and allow a labeling such that the two port numbers at both endpoints of any edge coincide. We therefore speak of the label of an edge and assume the set of labels to be $\{0, 1, 2\}$.

Moreover, we call the sequence of labels $l_0, l_1, l_2, \ldots$ of the edges traversed by an agent in a 3-regular graph G starting at a vertex v_0 a *traversal sequence* and say that the agent *follows* the traversal sequence $l_0, l_1, l_2, \ldots$ in G starting in v_0. Note that traversal sequences specify absolute labels to follow, whereas exploration sequences give offsets to the previous label in each step.

The most important building block for our construction are *barriers*. Intuitively, a barrier is a subgraph that cannot be crossed by a subset of the given set of agents. To define barriers formally, we need to describe how to connect two 3-regular graphs. Let B be a 3-regular graph with two distinguished edges $\{u, v\}$ and $\{u', v'\}$ both labeled 0, as shown in Figure 3. An arbitrary 3-regular graph G with at least two edges labeled 0 can be connected to B as follows: We remove the edges $\{u, v\}$ and $\{u', v'\}$ from B and two edges labeled 0 from G. We then connect each vertex of degree 2 in G with a vertex of degree 2 in B via an edge labeled 0. The vertices u, v, u', v' are referred to as *boundary vertices* of B, whereas all other vertices of B are called *interior vertices*. Any edge e with $e \neq \{u, v\}$ and $e \neq \{u', v'\}$ is referred to as *interior edge*.

Definition 4.1 (r-barrier). For $1 \leq r \leq k$, the graph B with distinguished edges $\{u, v\}$, $\{u', v'\}$ is an *r-barrier* for a set of k s-state agents $\mathcal{A}$ if for all graphs G connected to B as above, the following two properties hold:

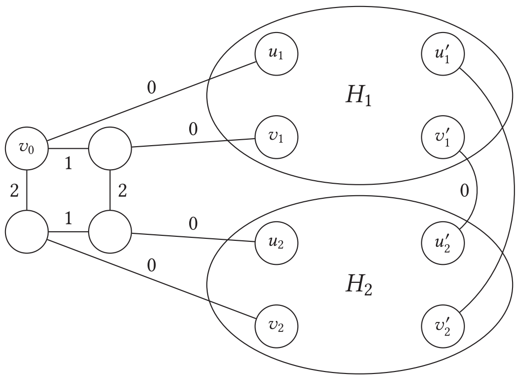


Fig. 4. Constructing a trap given two k-barriers H_1 and H_2.

(1) For all subsets of agents $\mathcal{A}' \subseteq \mathcal{A}$ with $|\mathcal{A}'| \leq r$ and every pair (a, b) in $\{u, v\} \times \{u', v'\}$ the following holds: If initially all agents $\mathcal{A}$ are at vertices of G, then no agent in the set $\mathcal{A}'$ can traverse B from a to b or vice versa when only agents in $\mathcal{A}'$ enter the subgraph B at any time during the traversal. We equivalently say that no subset of at most r agents can traverse B from a to b or vice versa.
(2) For all subsets of agents $\mathcal{A}' \subseteq \mathcal{A}$ with $|\mathcal{A}'| = r + 1$, if initially all agents in $\mathcal{A}$ are at vertices of G and agents in $\mathcal{A}'$ only enter B either via u and v, or via u' and v', then all agents in $\mathcal{A}'$ leave B either via u and v or via u' and v' if no other agents visit B during this traversal. In other words, if the set $\mathcal{A}'$ of agents enters B via the same distinguished edge, then it cannot split up such that a part of the agents leaves B via u or v and the other part via u' or v'.

A k-barrier immediately yields a trap for a set of agents.

Lemma 4.2. *Given a k-barrier with n vertices for a set of k agents $\mathcal{A}$, we can construct a trap with $2n + 4$ vertices for $\mathcal{A}$.*

Proof. Let H_1 and H_2 be two copies of a k-barrier for the set of agents $\mathcal{A}$ with distinguished edges $\{u_i, v_i\}$, $\{u'_i, v'_i\}$ of H_i. We connect the two graphs and four additional vertices, as shown in Figure 4. If the agents start in the vertex v_0, then none of the agents can reach u'_1 or v'_1 via the k-barrier H_1 or via the k-barrier H_2. Thus the agents $\mathcal{A}$ do not explore the graph. The constructed trap for the set of agents $\mathcal{A}$ contains $2n + 4$ vertices. □

Our goal for the remainder of the section is to construct a k-barrier for a given set of k agents $\mathcal{A}$ and to give a good upper bound on the number of vertices it contains. This will give an upper bound on the number of vertices of a trap by Lemma 4.2. The construction of the k-barrier is recursive. We start with a 1-barrier, which builds on the following useful result by Fraigniaud et al. [24] stating that, for any set of non-cooperative agents, there is a graph containing an edge that is not traversed by any of them. A set of agents is *non-cooperative* if the transition function δ_i of every agent A_i is completely independent of the state and location of the other agents, i.e., δ_i is independent of $\boldsymbol{\sigma}_{-i}$, see Section 2.2.3.

Theorem 4.3 ([24, Theorem 4]). *For any k non-cooperative s-state agents, there exists a 3-regular graph G on $O(ks)$ vertices with the following property: There are two edges $\{v_1, v_2\}$ and $\{v_3, v_4\}$ in G, the former labeled 0, such that none of the k agents traverses the edge $\{v_3, v_4\}$ when starting in v_1 or v_2.*

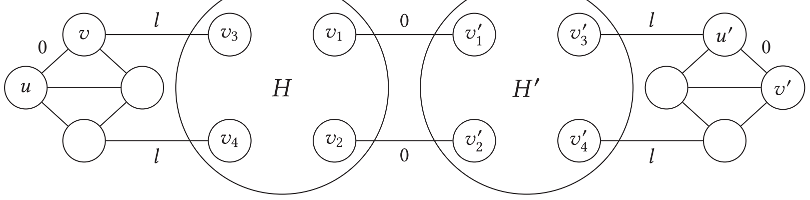


Fig. 5. A 1-barrier B for $\mathcal{A}$ for the case that $l \in \{1, 2\}$.

We proceed to generalize this construction for arbitrary starting states and collaborating agents.

Lemma 4.4. *For every set of k collaborating s-state agents $\mathcal{A} = \{A_1, \ldots, A_k\}$, there exists a 1-barrier B with $O(ks^2)$ vertices. Moreover, B remains a 1-barrier even if for all $i \in \{1, \ldots, k\}$ agent A_i with the set of states Σ_i starts in an arbitrary state $\sigma \in \Sigma_i$ instead of the starting state σ_i^*.*

Proof. Let $\mathcal{A} = \{A_1, \ldots, A_k\}$, let Σ_i be the set of states of A_i and let σ_i^* be its starting state. For all $i \in \{1, \ldots, k\}$ and all $\sigma \in \Sigma_i$, we define agent $A_i^{(\sigma)}$ to be the agent with the same behavior as A_i, but starting in state σ instead of σ_i^*. That is, $A_i^{(\sigma)}$ has the same set of states Σ_i as A_i and it transitions according to the function δ_i of A_i. Moreover, let $S := \{A_i^{(\sigma)} \mid i \in \{1, \ldots, k\},\ \sigma \in \Sigma_i\}$.

Applying Theorem 4.3 for the set of agents S yields a graph H with an edge $\{v_1, v_2\}$ labeled 0 and an edge $\{v_3, v_4\}$ labeled $l \in \{0, 1, 2\}$ so that any agent $A_i^{(\sigma)}$ that starts in v_1 or v_2 does not traverse the edge $\{v_3, v_4\}$. Let B be the graph consisting of two connected copies of H and 8 additional vertices, as illustrated in Figure 5. The edges $\{v_1, v_2\}$ and $\{v_1', v_2'\}$ are replaced by $\{v_1, v_1'\}$ and $\{v_2, v_2'\}$, which are also labeled 0. The edges $\{v_3, v_4\}$ and $\{v_3', v_4'\}$ with label l are deleted and v_3 and v_4 are connected each to one of the two two-degree vertices of a diamond graph by an edge with label l. The same connection to a diamond graph is added for v_3' and v_4' as shown in Figure 5. The edge labels of the two diamond graphs are arbitrary. Since each diamond graph has two vertices of degree three, each diamond graph has at least one edge with label 0. We choose one edge with label 0 and call the end vertices u and v (resp. u', v'). Note that in Figure 5, we have $l \in \{1, 2\}$; for the case that $l = 0$ the edge $\{u, v\}$ is the unique edge between the two vertices that are not adjacent to v_3 or v_4.

We claim that B is a 1-barrier for $\mathcal{A}$ with the distinguished edges $\{u, v\}$ and $\{u', v'\}$. Assume for the sake of contradiction, that the first property does not hold, i.e., there is a graph G that can be connected to B via the pairs of vertices $\{u, v\}$ and $\{u', v'\}$ so that if the agents $\mathcal{A}$ start in G in an arbitrary state, there is an agent A_j that walks (without loss of generality) from u to u' in B while there are no other agents in B. Then, A_j in particular walks from v_1' or v_2' to v_3' or v_4' in H' and starts this walk in a state $\sigma \in \Sigma_j$. But the traversal sequence of A_j in H' is the same as that of $A_j^{(\sigma)}$ that starts at v_1' or v_2'. This would imply that $A_i^{(\sigma)}$ traverses the edge $\{v_3, v_4\}$ in the original graph H when starting in v_1 or v_2, which contradicts Theorem 4.3.

To prove the second property of a 1-barrier, assume that there is a set of two agents, such that both enter B via the same distinguished edge without the other agents entering B and one of them exits B via u or v and the other via u' or v'. But then again one of the agents must have traversed H alone starting in v_1 or v_2 in a state σ and finally traversed the edge with label l incident to v_3 or v_4 or similarly in H' with v_1', v_2', v_3', v_4'. This leads to the same contradiction as above.

The whole proof does not use the specific starting states of the agents $\mathcal{A}$ and, in particular, the definition of S is independent of the starting states of the agents. Consequently, B is a 1-barrier for $\mathcal{A}$ even if we change the starting states of the agents.

Since every agent has s states, we obtain that the cardinality of S is bounded by $O(ks)$ and, hence, the graph B has $O(ks^2)$ vertices by Theorem 4.3. □

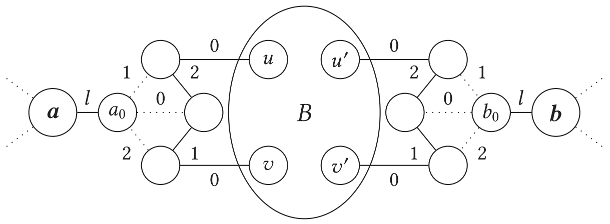


Fig. 6. An edge $\{a, b\}$ labeled l is replaced with the gadget $B(l)$ containing an r-barrier B. Only the dotted edges incident to a_0 and b_0 that are not labeled l are part of the gadget. Consequently, the gadget contains two vertices of degree 2. The vertices a and b are macro vertices of the graph $G(B)$.

The proof of Theorem 4.3 in Reference [24] uses the fact that when traversing a 3-regular graph the next state of an s-state agent only depends on the previous state and the label $l \in \{0, 1, 2\}$ of the edge leading back to the previous vertex. Thus, after at most $3s$ steps, the state of the agent and therefore also the next label chosen need to repeat with a period of length at most $3s$. For cooperative agents, however, the next state and label that are chosen may also depend on the positions and states of the other agents. We therefore need to account for the positions of all agents when forcing them into a periodic behavior. To this end, we will consider the relative positions of the agents with respect to a given vertex v. For our purposes, it is sufficient to define the relative position of an agent A_i by the shortest traversal sequence leading from v to the location of A_i. This motivates the following definition.

Definition 4.5. The *configuration* of a set of k agents $\mathcal{A} = \{A_1, \ldots, A_k\}$ in a graph G *with respect to a vertex* v is a $(3k)$-tuple $(\sigma_1, l_1, r_1, \sigma_2, l_2, r_2, \ldots, \sigma_k, l_k, r_k)$, where σ_i is the current state of A_i, l_i is the label of the edge leading back to the previous vertex visited by A_i and r_i is the shortest traversal sequence from v to A_i, where ties are broken in favor of lexicographically smaller sequences and where we set $r_i = \perp$ if the location of A_i is v.

To limit the number of possible configurations, we will force the agents to stay close together. Intuitively, we can achieve this for any graph G by replacing all edges with $(k-1)$-barriers. This way, only all agents together can move between neighboring vertices of the original graph G. To formalize this, we first need to explain how edges of a graph can be replaced by barriers. Since our construction may not be 3-regular, we need a way to extend it to a 3-regular graph.

Definition 4.6. Given a graph G, with vertices of degrees 2 and 3, we define the *3-regular extension* $\overline{G}$ as the graph resulting from copying G and connecting every vertex v of degree 2 to its copy v'. As the edges incident to v and v' have the same labels, it is possible to label the new edge $\{v, v'\}$ with a locally unique label in $\{0, 1, 2\}$.

Note that the 3-regular extension only increases the number of vertices of the graph by a factor of 2. Given a 3-regular graph G with distinguished edges e_1, e_2 labeled 0 and an r-barrier B for a set of k agents $\mathcal{A}$ with $k \geq r$, we replace all edges of G except for e_1 and e_2 using the following construction. First, for every $l \in \{0, 1, 2\}$, we replace every edge $\{a, b\}$ labeled l (except for the distinguished edges e_1 and e_2) with the gadget $B(l)$ shown in Figure 6, and we call the resulting graph $G^0(B)$. By construction, the labels of the edges incident to the same vertex in $G^0(B)$ are distinct. However, certain vertices only have degree 2. We take the 3-regular extension of $G^0(B)$ and define the resulting graph as $G(B) := \overline{G^0(B)}$.

The graph $G(B)$ contains two copies of $G^0(B)$. To simplify exposition, we identify each vertex v with its copy v' in $G(B)$. Then, there is a canonical bijection between the vertices in G and the vertices in $G(B)$, which are not part of a gadget $B(l)$. These vertices can be thought of as the original vertices of G, and we call them *macro vertices*.

We further connect a graph $G(B)$ to an arbitrary 3-regular graph G' as follows: Let $e_1 = \{u_1, v_1\}$ and $e_2 = \{u_2, v_2\}$ be the two distinguished edges of G with label 0. We remove the edges e_1, e_2 from $G(B)$ and also two edges of G' with label 0. Then, we connect each of the vertices u_1, v_1, u_2, v_2 with one vertex of G' of degree 2. We will use this construction in the recursive construction of barriers, because, as shown later, for a suitable graph G, the graph $G(B)$ is an $(r + 1)$-barrier with distinguished edges e_1, e_2.

In the following, we show several results about graphs of the form $G(B)$ connected to an arbitrary 3-regular graph G' as outlined above. We sometimes omit specifying the exact distinguished edges of G if these can be chosen arbitrarily. Moreover, we say that a group of agents $\mathcal{A}$ is *moving in the interior of* $G(B)$ if all agents only visit interior vertices of $G(B)$, i.e., they do not visit any of the boundary vertices $\{u_1, v_1, u_2, v_2\}$. The vertices of $G(B)$ corresponding to vertices of G other than $\{u_1, v_1, u_2, v_2\}$ are called the *interior macro vertices*, whereas $\{u_1, v_1, u_2, v_2\}$ are referred to as *boundary macro vertices*. Recall that we call any edge e in G with $e \neq e_1$ and $e \neq e_2$ an *interior edge*.

We now establish that the agents always stay close to each other in the graph $G(B)$.

Lemma 4.7. *Let G, G' be two connected 3-regular graphs and let B be a $(k-1)$-barrier for a set of k agents $\mathcal{A}$ with s states each. If the agents $\mathcal{A}$ start at arbitrary vertices of G' and then traverse the graph resulting from connecting $G(B)$ to G', then the following statements hold:*

(1) For all interior edges $\{v, v'\}$ in G, no strict subset $\mathcal{A}' \subsetneq \mathcal{A}$ of the agents can get from macro vertex v to macro vertex v' in $G(B)$ by traversing the gadget $B(l)$ connecting v and v' (where $l \in \{0, 1, 2\}$) without all other agents also entering this gadget.

(2) If the macro vertex v in $G(B)$ most recently visited by an agent in $\mathcal{A}$ is an interior vertex, then all agents are at v or in the surrounding gadgets $B(0)$, $B(1)$, and $B(2)$.

Proof. For the sake of contradiction, assume that there is a strict subset of agents $\mathcal{A}' \subsetneq \mathcal{A}$ that walks from a macro vertex v in $G(B)$ via the gadget $B(l)$ (where $l \in \{0, 1, 2\}$) to a distinct macro vertex v' without all other agents also entering this gadget connecting v and v' at any time during the traversal. The graph $G(B)$ contains two copies of $G^0(B)$, but all vertices in the $(k-1)$-barriers within $G^0(B)$ have degree 3. Thus, $\mathcal{A}'$ must have traversed some $(k-1)$-barrier B while only agents in $\mathcal{A}'$ enter B at any time of the traversal. This is a contradiction, as $|\mathcal{A}'| \leq k - 1$ and B is a $(k-1)$-barrier. Thus, for any agent in $\mathcal{A}$ to get from the macro vertex v to the distinct macro vertex v' via the gadget $B(l)$ connecting v and v', all k agents $\mathcal{A}$ need to enter the gadget $B(l)$ during the traversal. This shows the first claim.

For the second part of the claim, note that because of Property 2 for the barrier B the agents cannot split up into two groups such that after the traversal of the gadget connecting v and v' one group is at v (or one of the vertices at distance at most 4 from v that are not part of the barrier B) and the other group is at v' (or one of the vertices at distance 4 from v' that are not part of the barrier B). By the first part of the proof, all agents have to enter a gadget $B(l)$ with $l \in \{0, 1, 2\}$ to reach an interior macro vertex of $G(B)$ after starting in G'. By Property 2, the agents $\mathcal{A}$ cannot split up while only visiting interior vertices of $G(B)$. Hence, if v is the macro vertex last visited by an agent in $\mathcal{A}$ and v is an interior macro vertex (this means that, in particular, none of the agents visited a vertex of G' after visiting v), then all agents must be located at v or in the surrounding gadgets. □

Let B be a $(k-1)$-barrier for a set of k cooperative s-state agents $\mathcal{A} = \{A_1, \ldots, A_k\}$. We will frequently consider the configuration of $\mathcal{A}$ with respect to some macro vertex v in a graph of the form $G(B)$. Recall from the definition that the graph $G(B)$ contains two copies of the graph $G^0(B)$ and actually there exists a macro vertex v and a copy v'. Thus, when we talk about configurations of $\mathcal{A}$ in $G(B)$ with respect to some macro vertex v, we mean that we consistently choose one of the copies $G^0(B)$ and consider the configuration of $\mathcal{A}$ with respect to the macro vertex in this copy.

The behavior of a single agent A in a 3-regular graph is rather simple. If after t_1 steps in a 3-regular graph G_1 the state of A and incoming port number is the same as after t_2 steps in a 3-regular graph G_2, then, in both cases, the agent does the same state transition and chooses the edge with the same label. This means that, first of all, in one 3-regular graph (i.e., $G_1 = G_2$) the behavior of the agent quickly becomes periodic, and secondly, the agent has exactly the same behavior in every 3-regular graph. In particular, the traversal sequence of one agent A is the same in every 3-regular graph. The intuitive reason is that the agent can gain no new information while traversing a 3-regular graph, because these graphs locally look the same.

We want to obtain a similar result for a set of agents. However, in general, it is not true that if the configuration of a set of agents in a graph G_1 with respect to a vertex v_1 after t_1 steps is the same as after t_2 steps in G_2 with respect to a vertex v_2, then the next configurations and chosen labels of each agent coincide. This is because an agent can be used to mark a particular vertex and this can be used to detect differences in two 3-regular graphs G_1 and G_2 (or differences in the local neighborhood of v_1 and v_2 for $G_1 = G_2$). For instance, one agent could remain at a certain vertex while the other one walks in a loop that is only part of one of the graphs, but not the other. This may then lead to different configurations. That is why we consider graphs of the form $G(B)$. In these graphs, all interior macro vertices look the same, as they are surrounded by the same gadgets, and the agents have to stay close together, making it impossible for the agents to detect a loop that is part of one of the graphs, but not the other. This intuition is formally expressed in the following technical lemma.

Lemma 4.8. *Let B be a $(k-1)$-barrier for a set of k s-state agents $\mathcal{A}$, and let G_1, G_2, G'_1 and G'_2 be 3-regular graphs. Assume that, for $i \in \{1,2\}$, the graph $G_i(B)$ is connected to G'_i and the agents $\mathcal{A}$ start at arbitrary vertices of G'_i. Further, consider the configuration of $\mathcal{A}$ after t_i exploration steps where the last macro vertex v_i visited is an interior macro vertex of $G_i(B)$. If the configuration after t_1 steps with respect to v_1 is the same as after t_2 with respect to v_2, then one of the following claims holds:*

- *For $i \in \{1,2\}$, the agents only visit the last macro vertex v_i or vertices of the surrounding gadgets for the remainder of the exploration of the respective graph.*
- *There is $l \in \{0,1,2\}$ such that in both graphs the agents traverse the gadget $B(l)$ to the next macro vertex w_i in $G_i(B)$. Moreover, the configuration of the agents $\mathcal{A}$ with respect to w_1 when the first agent visits w_1 is the same as the configuration with respect to w_2 when the first agent visits w_2.*

Proof. The graphs $G_1(B)$ and $G_2(B)$ locally look the same to the agents, since the macro vertices v_1 and v_2 are surrounded by the same gadgets, as shown in Figure 7. Formally, this means that there is a canonical graph isomorphism γ from the induced subgraph of $G_1(B)$ containing v_1, all surrounding gadgets and the neighboring macro vertices to the induced subgraph of $G_2(B)$ containing v_2, all surrounding gadgets and the neighboring macro vertices. Moreover, γ respects the labeling and maps v_1 to v_2. Note that it is important that both v_1 and v_2 are interior macro vertices for the isomorphism to exist.

As the configuration of $\mathcal{A}$ after t_1 steps with respect to v_1 is the same as the configuration of $\mathcal{A}$ after t_2 steps with respect to v_2, the isomorphism also respects the positions of all the agents. By Lemma 4.7, we further know for $i \in \{1,2\}$ that, as long as v_i in $G_i(B)$ is the last macro vertex visited

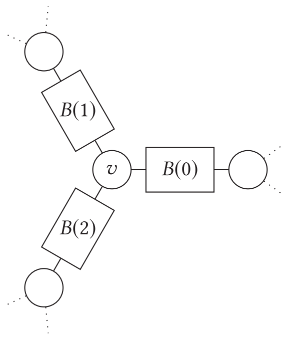


Fig. 7. An interior macro vertex v in a graph $G(B)$ surrounded by the three gadgets $B(0), B(1)$ and $B(2)$.

by the agents $\mathcal{A}$, all agents are at v_i or the surrounding gadgets. Iteratively, for $h = 0, 1, \ldots$ the following holds until the agents reach a macro vertex distinct from v_i in $G_i(B)$:

(1) For every agent $A \in \mathcal{A}$, the state of A and the incoming port number after $t_1 + h$ steps in $G_1(B)$ is the same as the state of A and the incoming port number after $t_2 + h$ steps in $G_2(B)$.
(2) The isomorphism γ maps the position of every agent $A \in \mathcal{A}$ after $t_1 + h$ steps in $G_1(B)$ to the position of A after $t_2 + h$ steps in $G_2(B)$.

In particular, this means that if the agents in $G_1(B)$ never visit a macro vertex distinct from v_1 after step t_1, then also the agents never visit a macro vertex distinct from v_2 after step t_2 in $G_2(B)$. However, if after $t_1 + \bar{h}$ steps for some $\bar{h} \in \mathbb{N}$ one agent $A \in \mathcal{A}$ first visits the distinct macro vertex w_1 in $G_1(B)$, then after $t_2 + \bar{h}$ steps in $G_2(B)$ agent A also visits the distinct macro vertex w_2 for the first time. At this moment, the configuration of $\mathcal{A}$ with respect to v_1 is the same as the configuration of $\mathcal{A}$ with respect to v_2. This implies that v_1 and w_1 are connected with the same gadget $B(l)$ as v_2 and w_2. Furthermore, also the configuration of $\mathcal{A}$ with respect to w_1 after $t_1 + \bar{h}$ steps must be the same as the configuration of $\mathcal{A}$ with respect to w_2 after $t_2 + \bar{h}$ steps. $\square$

Let G, G' be two 3-regular graphs and B be a $(k-1)$-barrier for a set of k agents $\mathcal{A}$ with s states each. While traversing the graph $G(B)$ connected to G', assume that the agents $\mathcal{A}$ in step t_0 first visit an interior macro vertex v_0 distinct from the previous macro vertex visited by any agent in $\mathcal{A}$. Further, let x_0 be the configuration of $\mathcal{A}$ in step t_0 with respect to v_0. Iteratively, for $i > 0$, define t_i to be the first point in time after t_{i-1}, when one of the agents in $\mathcal{A}$ visits an interior macro vertex v_i distinct from v_{i-1}. We also say that $\mathcal{A}$ *arrives at* v_i at this exploration step. Note that as soon as $\mathcal{A}$ visit a boundary vertex of $G(B)$, we abort and the sequence ends. The vertex v_i is a neighbor of v_{i-1} in G and, by Lemma 4.7, all agents are at v_i or the surrounding gadgets. The sequence of macro vertices $v_0, v_1, \ldots$, which is a sequence of neighboring vertices in G, yields a unique sequence of labels $l_0, l_1, \ldots$ of the edges between the neighboring vertices in G, which we call the *macro traversal sequence* of $\mathcal{A}$ starting in vertex v_0 of $G(B)$ in configuration x_0. Note that the macro traversal sequence may be finite if the agents visit a boundary macro vertex or stop exploring distinct macro vertices. From Lemma 4.8, we obtain that the configuration x_0 in step t_0 completely determines all labels of the macro label sequence independent of the underlying graph G (the graph G may, however, influence when the macro label sequence terminates, because the agents $\mathcal{A}$ visit a boundary vertex).

Before we can present the recursive construction of barriers, we need to introduce an additional definition. Let $k, r \in \mathbb{N}$ be such that $2 \leq r \leq k$. To construct an r-barrier B' for a set $\mathcal{A}$ of k cooperative s-state agents given an $(r-1)$-barrier B, we need to examine the behavior of all subsets of r agents. There are $\binom{k}{r}$ subsets of r agents and the behavior of two different subsets of r agents may be completely different. We denote these $\binom{k}{r}$ subsets of r agents by $\mathcal{A}_1^{(r)}, \ldots, \mathcal{A}_{\binom{k}{r}}^{(r)}$.

Assume that we have an $(r-1)$-barrier B for a set of k agents $\mathcal{A}$ and let G, G' be two 3-regular graphs such that $G(B)$ is connected to G'. We assume that the agents $\mathcal{A}$ start at arbitrary vertices of G'. For $1 \leq j \leq \binom{k}{r}$, consider the situation that only the subset of agents $\mathcal{A}_j^{(r)}$ enters the subgraph $G(B)$ and let v be the last interior macro vertex visited by the agents $\mathcal{A}_j^{(r)}$. Until these agents visit a distinct macro vertex w, all agents in $\mathcal{A}_j^{(r)}$ are located at v or the surrounding gadgets $B(0), B(1), B(2)$ by Lemma 4.7. Thus, the number of possible locations of the agents can be bounded in terms of the size of the gadgets $B(0)$, $B(1)$, and $B(2)$. In addition, every agent has at most s states. Therefore the number of configurations of $\mathcal{A}_j^{(r)}$ with respect to the macro vertex v last visited is finite and can be bounded in terms of s and the size of the gadgets. We define α_B to be the number of possible configurations of $\mathcal{A}_j^{(r)}$ with respect to an interior macro vertex v of $G(B)$ in the exploration step when the agents $\mathcal{A}_j^{(r)}$ arrive at v, i.e., some agent in $\mathcal{A}_j^{(r)}$ first visits v. Note that α_B is a bound on the number of possible configurations and hence is independent of the specific subset of agents $\mathcal{A}_j^{(r)}$. As the local neighborhood of v, i.e., the three gadgets surrounding v, does not depend on the graph G, the definition also does not depend on the 3-regular graph G.

Given the definition of α_B, we are now in a position to present the construction of an r-barrier given an $(r-1)$-barrier. We will later bound α_B and, thus, the size of the r-barrier in Lemma 4.11.

Theorem 4.9. *Given an $(r-1)$-barrier B with n vertices for a set $\mathcal{A}$ of k agents with s states each, we can construct an r-barrier B' for $\mathcal{A}$ with the following properties:*

(1) *We have $B' = H(B)$ for a suitable 3-regular graph H with distinguished edges $e_1 = \{u_1, v_1\}$ and $e_2 = \{u_2, v_2\}$ labeled 0.*
(2) *Any path from u_1 or v_1 to u_2 or v_2 in B' contains at least 3 distinct barriers B.*
(3) *The r-barrier B' contains at most $O(\binom{k}{r} \cdot n \cdot \alpha_B^2)$ vertices.*

Proof. Let G, G' be two arbitrary 3-regular graphs and e, e' be two distinguished edges in G with label 0 such that $G(B)$ is connected to G' via the vertices incident to the distinguished edges. For $j \in \{1, 2, \ldots, \binom{k}{r}\}$, consider the subset of r agents $\mathcal{A}_j^{(r)}$ starting at arbitrary vertices of G'. By definition and Lemma 4.7, there are at most α_B possible configurations of the agents $\mathcal{A}_j^{(r)}$ whenever one of the agents in α_B first visits a new distinct interior macro vertex in $G(B)$. We can hence denote these possible configurations by $x_1, \ldots, x_{\alpha_B}$.

Assume that after t exploration steps an agent in $\mathcal{A}_j^{(r)}$ first visits an interior macro vertex v in $G(B)$ distinct from the previous macro vertex visited by any agent in $\mathcal{A}_j^{(r)}$. Moreover, let x_h for $h \in \{1, \ldots, \alpha_B\}$ be the configuration of $\mathcal{A}_j^{(r)}$ with respect to v at this time. By Lemma 4.8, the following holds: Either the agents $\mathcal{A}_j^{(r)}$ do not visit any macro vertex distinct from v after step t or x_h uniquely determines $l \in \{0, 1, 2\}$ such that the agents traverse the gadget $B(l)$ to the next macro vertex v' visited in $G(B)$ (this means that l only depends on x_h, $\mathcal{A}_j^{(r)}$ and B, but not on G).

We can therefore define a single agent $\bar{A}_j$ as follows: The set of states of $\bar{A}_j$ is $\{\sigma_1, \ldots, \sigma_{\alpha_B}\}$. Moreover, in state σ_h the agent $\bar{A}_j$ traverses the edge labeled l and transitions to $\sigma_{h'}$ if the set of agents $\mathcal{A}_j^{(r)}$ in configuration x_h at a time t traverses the gadget $B(l)$ to the next macro vertex v'. Here the configuration of $\bar{A}_j$ with respect to v' when the first agent visits v' is $x_{h'}$. If the agents $\bar{A}_j$ do not visit any macro vertex after visiting v in step t, then $\bar{A}_j$ terminates in state σ_h. The starting state of $\bar{A}_j$ corresponds to the configuration with respect to a vertex v, where all the agents in $\mathcal{A}_j^{(r)}$ are in their starting states and located at vertex v. Note that the transition function $\bar{\delta}$ of $\bar{A}_j$ described above is well-defined, because, by Lemma 4.8, the label l only depends on the configuration of $\mathcal{A}_j^{(r)}$ at t and is independent of the underlying graph G. By construction, there is

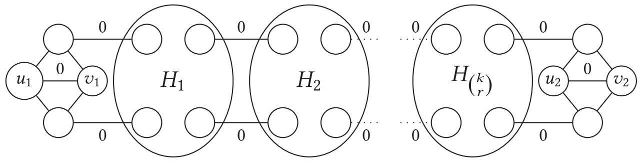


Fig. 8. Connecting the graphs $H_1, H_2, \ldots, H_{\binom{k}{r}}$ to a graph H, yields the r-barrier $H(B)$.

a one-to-one correspondence between the macro traversal sequence of $\mathcal{A}_j^{(r)}$ starting in $G(B)$ in an interior macro vertex v in a configuration x_h and the traversal sequence of agent $\bar{A}_j$ starting in the corresponding vertex v in G in state σ_h (as long as $\mathcal{A}_j^{(r)}$ does not visit any boundary vertex in $G(B)$).

Applying Lemma 4.4 for the single agent $\bar{A}_j$ yields a 1-barrier H_j with $O(\alpha_B^2)$ vertices that cannot be traversed by $\bar{A}_j$, irrespective of its starting state. We now connect the graphs $H_1, \ldots, H_{\binom{k}{r}}$ as shown in Figure 8, and we let H denote the resulting graph. We first show that the graph $B' := H(B)$ resulting from replacing all edges except for $e_1 = \{u_1, v_1\}$ and $e_2 = \{u_2, v_2\}$ by the barrier B is an r-barrier for $\mathcal{A}$. Afterwards, we show the three additional properties in the claim.

For the first property of an r-barrier, assume, for the sake of contradiction, that there is a subset of r agents $\mathcal{A}_j^{(r)}$ and some 3-regular graph G connected to $H(B)$ such that without loss of generality the agents $\mathcal{A}_j^{(r)}$ can traverse $H(B)$ from u_1 to u_2. Then there must be a consecutive subsequence $w_0, w_1, \ldots, w_h$ of the macro vertex sequence of $\mathcal{A}_j^{(r)}$ during the traversal of $H(B)$ with the following properties: The vertices $w_1, \ldots, w_{h-1}$ are contained in $H_j(B)$, w_0 and w_h are not contained in $H_j(B)$, w_1 and w_{h-1} (as vertices in the 1-barrier H_j) are incident to different distinguished edges (i.e., $\{u, v\}$ or $\{u', v'\}$ in Figure 5) of the 1-barrier H_j. Thus, the set of agents $\mathcal{A}_j^{(r)}$ starting in w_0 or the surrounding gadgets in a suitable configuration x_i with respect to w_0 traverses the graph $H_j(B)$ from w_1 to w_{h-1}. This means that for a suitable graph G' connected to H_j and starting state σ_i the agent $\bar{A}_j$ can traverse H_j. But this is a contradiction, as we constructed H_j as a 1-barrier for $\bar{A}_j$ using Lemma 4.4 and the 1-barrier H_j is independent of the starting state of $\bar{A}_j$.

For the second property of an r-barrier, let $\mathcal{A}' \subseteq \mathcal{A}$ be a set of agents with $|\mathcal{A}'| = r + 1$. Assume, for the sake of contradiction, that there is some graph G connected to $H(B)$ such that after the agents $\mathcal{A}'$ (and no other agents) enter $H(B)$ via u_1 and v_1, or via u_2 and v_2, a subset $\emptyset \neq \mathcal{A}'_1 \subsetneq \mathcal{A}'$ leaves $H(B)$ via u_1 or v_1 and the other agents $\mathcal{A}'_2 := \mathcal{A}' \setminus \mathcal{A}'_1$ via u_2 or v_2. Since B is an $(r-1)$-barrier, no set of at most $r - 1$ agents can get from an interior macro vertex to a distinct interior macro vertex in $H(B)$. Thus, we must have $|\mathcal{A}'_1| \geq r$ or $|\mathcal{A}'_2| \geq r$. Without loss of generality, we assume that the first case occurs, which implies $|\mathcal{A}'_1| = r$ and $|\mathcal{A}'_2| = 1$. For the single agent in $\mathcal{A}'_2$ to leave $H(B)$ via u_2 or v_2 at least $r - 1$ agents from $\mathcal{A}'_1$ must be in a gadget adjacent to u_2 or v_2. But all these $r - 1$ agents afterwards leave $H(B)$ via u_1 or v_1 and they need the remaining agent in $\mathcal{A}'_1$ to even get to a distinct macro vertex. But then the set of r agents $\mathcal{A}'_1$ traverses the subgraphs $H_j(B)$ for all $j \in \{1, \ldots, \binom{k}{r}\}$, which again leads to a contradiction as in the proof for the first property (for j such that $\mathcal{A}'_1 = \mathcal{A}_j^{(r)}$).

Finally, we obviously have $B' = H(B)$ for a 3-regular graph H by construction and the second additional property follows from the fact that any path from u_1 or v_1 to u_2 or v_2 in H has length at least 3. Further, each H_j contains $O(\alpha_B^2)$ vertices and therefore H has at most $O(\binom{k}{r} \cdot \alpha_B^2)$ vertices. Since B has n vertices, the number of vertices of $B' = H(B)$ is at most $O(\binom{k}{r} \cdot n \cdot \alpha_B^2)$, where we use that H is 3-regular and therefore the number of edges of H that are replaced by a copy of B is 3/2 times the number of its vertices. □

We now fix a set of k agents $\mathcal{A}$ with s states each and let B_1 be the 1-barrier given by Lemma 4.4 and B_r for $1 < r \leq k$ be the r-barrier constructed recursively using Theorem 4.9. Moreover, we

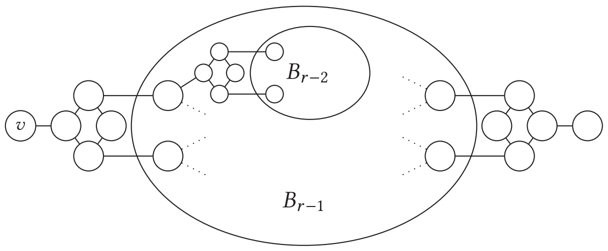


Fig. 9. Recursive structure of $B(l)$ containing i-barriers for $i \in \{1, \ldots, r-1\}$.

let n_r be the number of vertices of B_r and $\alpha_r := \alpha_{B_{r-1}}$ be the maximum number of possible configurations of a set of r agents with respect to an interior macro vertex in a graph of the form $G(B_{r-1})$.

We want to bound the number of vertices n_k of B_k and thus, according to Lemma 4.2, also the number of vertices of the trap for $\mathcal{A}$. By Theorem 4.9, there is a constant $c \in \mathbb{N}$ such that $n_r \leq c\binom{k}{r} n_{r-1}\alpha_r^2$. To bound n_r, we therefore need to bound α_r.

One possible way to obtain an upper bound on α_r is to use Lemma 4.7 stating that there always is a macro vertex v such that all agents are located at v or the surrounding gadgets. Counting the number of possible positions within these three gadgets and states of the agents then gives an upper bound on α_r. For the tight bound in our main result, however, we need a more careful analysis of the recursive structure of our construction and also need to consider the configurations of the agents at specific times. We start with the following definition and a technical lemma.

For $j \in \{1, \ldots, r-1\}$, we say that a vertex w' is *j-adjacent* to some other vertex w if there is a path P from w to w' that does not traverse a j-barrier B_j, i.e., P does not contain a subpath leading from one vertex of the distinguished edge $\{u, v\}$ to a vertex of the other distinguished edge $\{u', v'\}$ in B_j. As a convention, every vertex w is j-adjacent to itself for all $j \in \{1, \ldots, r-1\}$. Note that a vertex w' that is part of a j-barrier may be j-adjacent to some vertex w outside the barrier if there is a path from w to w' that does not traverse a distinct j-barrier.

Lemma 4.10. *Let G, G' be two 3-regular graphs such that $G(B_{r-1})$ is connected to G'. If v is an interior macro vertex in $G(B_{r-1})$, then for $j \in \{1, \ldots, r-1\}$ the number of vertices that are j-adjacent to v is bounded by $2^{4(r-j)}n_j$.*

Proof. To bound the number of j-adjacent vertices, we examine the recursive structure of one of the gadgets $B(l)$ incident to v, as shown in Figure 9. By Theorem 4.9, an $(r-1)$-barrier B' for $r \geq 3$ is constructed from a 3-regular graph H and an $(r-2)$-barrier B such that $B' = H(B)$. Hence, the gadget $B(l)$, which contains the barrier B_{r-1}, also contains many copies of the barrier B_{r-2}, which again contain many copies of the barrier B_{r-3} (if $r \geq 4$) and so on.

We first show that the distance from v to any j-adjacent vertex, which is not part of a barrier B_j and hence, in particular, not a boundary vertex of B_j, is at most $4(r-j)$. For $j = r-1$, consider Figure 6 showing how for $l \in \{0, 1, 2\}$ every interior edge $\{a, b\}$ with label l in G is replaced by the gadget $B_{r-1}(l)$ containing B_{r-1}. Note that actually $G(B_{r-1})$ contains a copy a' of a as well as a copy b' of b and there is one such gadget between a and b and another gadget between a' and b'. However, all vertices in both gadget, which are $(r-1)$-adjacent, but not in B_{r-1}, are at distance at most 4 from a for all $l \in \{0, 1, 2\}$ (the copy a_0' of a_0 is at distance 2 from a and every vertex in the copy is at distance at most 2 from a_0'). For $j = r-2, r-3, \ldots$ the claim follows by the same argument, since edges between two macro vertices in B_{r-1} are replaced by a barrier B_{r-2} and so on. See also the recursive structure given in Figure 9.

As $G(B_{r-1})$ is 3-regular, the number of vertices at distance at most $4(r-j)$ from v can be bounded by $3 \cdot 2^{4(r-j)-1}$. Hence there are at most $3 \cdot 2^{4(r-j)-1}$ vertices, which are j-adjacent but not part of a barrier B_j. Moreover, any j-barrier B_j containing vertices that are j-adjacent to v, in particular contains two vertices with a distance of at most $4(r-j)$ to v. This follows from the same analysis of the recursive structure of the barriers as above. As $G(B_{r-1})$ is 3-regular, there are at most $3 \cdot 2^{4(r-j)-1}$ vertices of distance at most $4(r-j)$ from v and therefore at most $3/2 \cdot 2^{4(r-j)-1}$ different j-barriers, with n_j vertices each, containing j-adjacent vertices. Thus, there are at most $3/2 \cdot 2^{4(r-j)-1} n_j$ vertices that are j-adjacent to v and part of a barrier B_j. Overall, the number of j-adjacent vertices to v can therefore be bounded by

$$3/2 \cdot 2^{4(r-j)-1} \cdot n_j + 3 \cdot 2^{4(r-j)-1} \leq 2^{4(r-j)} \cdot n_j,$$

where we used $n_j \geq 6$ and $j \leq r-1$. □

The idea now is to consider the configuration of the agents with respect to a macro vertex v_i exactly at the time t when at least $\lceil r/2 \rceil + 1$ agents are $\lceil r/2 \rceil$-adjacent to v_i. We then further use the fact that it is not possible to partition the agents $\mathcal{A}$ into two groups $\mathcal{A}'$ and $\mathcal{A}''$ with at most $i \geq \lceil r/2 \rceil$ agents each that are separated on any path by at least two i-barriers. This yields the following bound on α_r.

Lemma 4.11. *Let $\mathcal{A}$ be a set of k agents, $s \geq 2$ and $r \in \{2, \dots, k\}$. We then have*

$$\alpha_r \leq s^{7r^2} \cdot n_{\lceil r/2 \rceil}^{\lceil r/2 \rceil} \cdot n_{r-1} \cdot \prod_{j=\lceil r/2 \rceil + 1}^{r-1} n_j.$$

Proof. Let $\mathcal{A}^{(r)} \subseteq \mathcal{A}$ be an arbitrary subset of r agents and G, G' be two 3-regular graphs such that $G(B_{r-1})$ is connected to G'. Let further v be an interior macro vertex of $G(B_{r-1})$. We want to bound α_r, i.e., the number of configurations of $\mathcal{A}^{(r)}$ in the exploration step t when the agents arrive at v. Let v' be the last macro vertex of $G(B_{r-1})$ that was visited by one of the agents in $\mathcal{A}^{(r)}$ before.

Because of the recursive structure of the barriers, see Figure 9, every macro vertex is surrounded by $\lceil r/2 \rceil$-barriers and any path between the two macro vertices v' and v contains at least one barrier $B_{\lceil r/2 \rceil}$ (note that $r \geq 2$ by assumption). To reach the vertex v after visiting v', at least $\lceil r/2 \rceil + 1$ agents from $\mathcal{A}^{(r)}$ are necessary to traverse such an $\lceil r/2 \rceil$-barrier. Thus, at some step t_0 before the agents arrive at v at step t at least $\lceil r/2 \rceil + 1$ agents must be at a vertex that is $\lceil r/2 \rceil$-adjacent to v, as otherwise the agents would not be able to reach v. The crucial observation at this point is that the number of possible configurations in step t_0 also bounds the number α_r of possible configurations in step t, because, by Lemma 4.8, the configurations in step t must coincide if they already coincided in step t_0.

Let $\mathcal{A}_1$ denote the set of agents that are at a vertex that is $\lceil r/2 \rceil$-adjacent to v at time t_0 and let $\mathcal{A}_2 := \mathcal{A}^{(r)} \setminus \mathcal{A}_1$. By the argument above, we have $|\mathcal{A}_1| \geq \lceil r/2 \rceil + 1$. We claim the following: For $j \in \{1, \dots, |\mathcal{A}_2|\}$, there are at least $(r-j)$ agents in $\mathcal{A}^{(r)}$ that are located at a vertex that is $(r-j)$-adjacent to v.

For $j = |\mathcal{A}_2|$, we have $r - j = |\mathcal{A}_1| > \lceil r/2 \rceil$. Thus, the claim holds by definition of $\mathcal{A}_1$, since there are $r-j$ agents, namely the set of agents $\mathcal{A}_1$, which are located at vertices that are $\lceil r/2 \rceil$-adjacent to v and thus also $(r-j)$-adjacent to v, because $r - j > \lceil r/2 \rceil$.

Now, assume for the sake of contradiction that the claim holds for j, but not for $j-1$. This means that there is a subset of agents $\mathcal{A}' \subset \mathcal{A}^{(r)}$ with $|\mathcal{A}'| = r - j$ such that all agents in $\mathcal{A}'$ are located at vertices that are $(r-j)$-adjacent to v. But for $j-1$ the claim does not hold, which implies that all other agents $\mathcal{A}'' := \mathcal{A}^{(r)} \setminus \mathcal{A}'$ are at vertices that are not $(r-j+1)$-adjacent: If there was an agent $A \in \mathcal{A}''$ at a vertex that is $(r-j+1)$-adjacent, then $\mathcal{A}' \cup \{A\}$ would be a set of $(r-j+1)$ agents that are all at $(r-j+1)$-adjacent vertices, which is a contradiction to the choice of j.

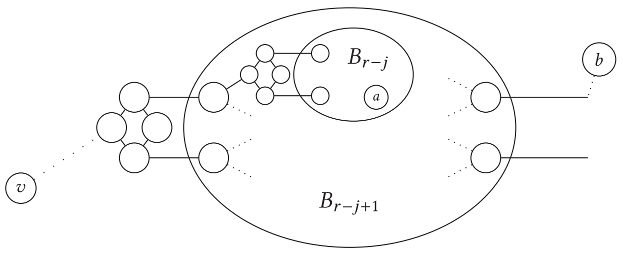


Fig. 10. Vertex a is $(r-j)$-adjacent to v, while b is not $(r-j+1)$-adjacent to v.

But the path between any pair of vertices (a, b), such that a is $(r-j)$-adjacent to v and b is not $(r-j+1)$-adjacent to v, contains at least two $(r-j)$-barriers, see also Figure 10. The reason is that $r-j+1 > \lceil r/2 \rceil \geq 1$ and, by Theorem 4.9, any path from u or v to u' or v' contains at least three $(r-j)$ barriers. Thus the set of agents $\mathcal{A}'$ and $\mathcal{A}''$ are separated by at least two $(r-j)$-barriers on any path and $|\mathcal{A}'| \leq r-j$ as well as $|\mathcal{A}''| = j < r-j$, since $j \leq \lceil r/2 \rceil - 1$. But then a set of at most $r-j$ agents must have traversed a barrier B_{r-j} or a set of at most $r-j-1$ agents must have traversed the gadget between two macro vertices in B_{r-j}, which both is a contradiction.

By the claim above, we can enumerate the agents in $\mathcal{A}^{(r)}$ as $A_1, A_2, \ldots, A_r$, so that:

(1) For $j \in \{1, \ldots, |\mathcal{A}_1|\}$, $A_j \in \mathcal{A}_1$ and the location of A_j is $\lceil r/2 \rceil$-adjacent to v.
(2) For $j \in \{|\mathcal{A}_1| + 1, \ldots, r-1\}$, $A_j \in \mathcal{A}_2$ and the location of A_j is j-adjacent to v.
(3) Agent $A_r \in \mathcal{A}_2$ is at v or one of the surrounding gadgets by Lemma 4.7.

We first bound the number of possible locations of the agents and afterwards consider the number of possible states and possible edge labels to the previous vertex.

There are $r!$ possible permutations of the agents. Moreover, using Lemma 4.10, we can bound the number of possible locations at time t_0 of the agents in $\mathcal{A}_1$ by $(2^{4(r-\lceil r/2 \rceil)} n_{\lceil r/2 \rceil})^{|\mathcal{A}_1|}$, the number of possible locations of the agents $\{A_{|\mathcal{A}_1|+1}, \ldots, A_{r-1}\}$ by $\prod_{j=|\mathcal{A}_1|+1}^{r-1} 2^{4(r-j)} n_j$ and the number of possible locations of A_r by $2^4 n_{r-1}$. Overall, we can thus bound the number of possible locations of the agents $\mathcal{A}^{(r)}$ at t_0 with respect to v by

$$r! \cdot \left(2^{4(r-\lceil r/2 \rceil)} n_{\lceil r/2 \rceil}\right)^{|\mathcal{A}_1|} \left(\prod_{j=|\mathcal{A}_1|+1}^{r-1} 2^{4(r-j)} n_j \right) 2^4 n_{r-1}$$
$$\leq r! \cdot (2^{4r})^r \cdot n_{\lceil r/2 \rceil}^{|\mathcal{A}_1|} \cdot n_{r-1} \cdot \prod_{j=|\mathcal{A}_1|+1}^{r-1} n_j \leq 2^{5r^2} \cdot n_{\lceil r/2 \rceil}^{\lceil r/2 \rceil} \cdot n_{r-1} \cdot \prod_{j=\lceil r/2 \rceil+1}^{r-1} n_j,$$

where we used $r! \leq r^r \leq 2^{r^2}$ and $n_{j-1} \leq n_j$ for all $j \in \{2, \ldots, r-1\}$.

To bound the number of configurations of the agents $\mathcal{A}^{(r)}$ note that there are s^r possible states of the agents and for each agent 3 possible edge labels to the previous vertex. Combining these bounds with the above bound on the number of locations of the agents, we obtain the following bound on the number of configurations of $\mathcal{A}^{(r)}$ at t_0 with respect to v:

$$s^r \cdot 3^r \cdot 2^{5r^2} \cdot n_{\lceil r/2 \rceil}^{\lceil r/2 \rceil} \cdot n_{r-1} \cdot \prod_{j=\lceil r/2 \rceil+1}^{r-1} n_j \leq s^{7r^2} \cdot n_{\lceil r/2 \rceil}^{\lceil r/2 \rceil} \cdot n_{r-1} \cdot \prod_{j=\lceil r/2 \rceil+1}^{r-1} n_j.$$

Here, we used $s \geq 2$ and $r \geq 2$. By the observation at the beginning of the proof, the number of possible configurations of $\mathcal{A}^{(r)}$ at t_0 with respect to v also bounds α_r. $\square$

Using the bound on α_r from Lemma 4.11, we can bound the number of vertices of the barriers.

Theorem 4.12. *For every set of k agents $\mathcal{A}$ with s states each and every $r \leq k$, there is an r-barrier with at most $O(s^{k \cdot 2^{4 \cdot r}})$ vertices.*

Proof. The existence of an r-barrier follows from Lemma 4.4 and Theorem 4.9, and we further have the following bound on the number of vertices n_r of B_r for a sufficiently large constant $c \in \mathbb{N}$:

$$n_1 \leq cks^2 \quad \text{and} \quad n_r \leq c\binom{k}{r} n_{r-1} \alpha_r^2.$$

It is without loss of generality to assume $s \geq 2$, since otherwise a trap of constant size can trivially be found. Hence, we can plug in the bound on α_r from Lemma 4.11. For the asymptotic bound, we may assume $c \leq s^k$, and we further have $\binom{k}{r} \leq 2^k$. We therefore get

$$\begin{aligned} n_r &\leq s^k \cdot 2^k \cdot n_{r-1} \cdot \left(s^{7 \cdot r^2}\right)^2 \cdot \left(n_{\lceil r/2 \rceil}^{\lceil r/2 \rceil}\right)^2 \cdot n_{r-1}^2 \prod_{j=\lceil r/2 \rceil+1}^{r-1} n_j^2 \\ &\leq s^{2k+14r^2} \cdot n_{\lceil r/2 \rceil}^{(r+1)} \cdot n_{r-1}^3 \prod_{j=\lceil r/2 \rceil+1}^{r-1} n_j^2. \end{aligned} \tag{4}$$

We proceed to show inductively that $n_r \leq s^{k \cdot 2^{4 \cdot r}}$ holds for all $r \in \{1, \ldots, k\}$. For $r = 1$, we have $n_1 \leq cks^2 \leq s^{4k} \leq s^{k \cdot 2^4}$. Let us assume the claim holds for $1, \ldots, r-1$. From Inequality Equation (4), we obtain

$$\begin{aligned} n_r &\leq s^{2k+14r^2} \cdot \left(s^{k \cdot 2^{4 \cdot \lceil r/2 \rceil}}\right)^{r+1} \cdot \left(s^{k \cdot 2^{4(r-1)}}\right)^3 \cdot \prod_{j=\lceil r/2 \rceil+1}^{r-1} \left(s^{k \cdot 2^{4 \cdot j}}\right)^2 \\ &= s^{2k+14r^2+k \cdot (r+1) \cdot 2^{4 \cdot \lceil r/2 \rceil}+3 \cdot k \cdot 2^{4(r-1)}+2k \sum_{j=\lceil r/2 \rceil+1}^{r-1} 2^{4 \cdot j}}. \end{aligned}$$

Thus, it is sufficient to bound the exponent. As $r \geq 2$, we have $\sum_{i=0}^{r-1} 2^{4 \cdot i} = (2^{4r} - 1)/(2^4 - 1) \leq 2 \cdot 2^{4(r-1)}$ as well as $(r+1) \cdot 2^{4\lceil r/2 \rceil} \leq 4 \cdot 2^{4(r-1)}$ and $2k + 14r^2 \leq 2 \cdot k \cdot 2^{4(r-1)}$. Hence, we obtain

$$\begin{aligned} &2k + 14r^2 + k \cdot (r+1) \cdot 2^{4 \cdot \lceil r/2 \rceil} + 3 \cdot k \cdot 2^{4(r-1)} + 2k \sum_{j=\lceil r/2 \rceil+1}^{r-1} 2^{4 \cdot j} \\ &\leq k \cdot \left(2 \cdot 2^{4(r-1)} + 4 \cdot 2^{4(r-1)} + 3 \cdot 2^{4(r-1)} + 4 \cdot 2^{4(r-1)}\right) \leq k \cdot 2^{4 \cdot r}. \end{aligned}$$

This shows $n_r \leq s^{k \cdot 2^{4r}}$, as desired. □

The bound for the barriers above immediately yields the bound for the trap for k agents.

Theorem 4.13. *For any set $\mathcal{A}$ of k agents with at most s states each, there is a trap with at most $O(s^{2^{5k}})$ vertices.*

Proof. We can always add additional unreachable states to all agents so that all of them have s states. Theorem 4.12 yields a k-barrier for a given set of k agents $\mathcal{A}$ with $O(s^{k \cdot 2^{4 \cdot k}})$ vertices. The claim follows from the fact that $k \cdot 2^{4 \cdot k} \leq 2^{5 \cdot k}$ and that a k-barrier with n vertices yields a trap with $O(n)$ vertices for $\mathcal{A}$ by Lemma 4.2. □

Finally, we derive a bound on the number of agents k that are needed for exploring every graph on at most n vertices.

Theorem 4.14. *The number of agents needed to explore every graph on at most n vertices is at least $\Omega(\log \log n)$, if we allow $O((\log n)^{1-\varepsilon})$ bits of memory for an arbitrary constant $\varepsilon > 0$ for every agent.*

Proof. Let $\mathcal{A}$ be a set of k agents with $O((\log n)^{1-\varepsilon})$ bits of memory that explores any graph on at most n vertices. By otherwise adding some unused memory, we may assume that $0 < \varepsilon < 1$ and that there is a constant $c \in \mathbb{N}$ such that all agents in $\mathcal{A}$ have $s := 2^{c\cdot(\log n)^{1-\varepsilon}}$ states. We apply Theorem 4.13 and obtain a trap for $\mathcal{A}$ containing $O(s^{2^{5\cdot k}})$ vertices. As the set of agents $\mathcal{A}$ explore any graph on at most n vertices, we have $n \le O(1)s^{2^{5\cdot k}}$. By taking logarithms on both sides of this inequality, we obtain

$$\log n \le O(1) + 2^{5k}\log s = O(1) + 2^{5k}\cdot c\cdot(\log n)^{1-\varepsilon}.$$

Multiplication by $(\log n)^{\varepsilon-1}$ on both sides and taking logarithms yields the claim. □

Observe that the entire argument eventually leading to Theorem 4.14 relies on bounding the number of different configurations of the system (in the sense of Lemma 4.11), and, in particular, on the number of steps until a configuration repeats (cf. Lemma 4.8 and Theorem 4.9). Clearly, the number of configurations is only smaller for a single agent with pebbles, since pebbles do not have a state or an incoming edge label associated with them. Our proof therefore carries over to this setting (for an explicit proof in this setting, we refer to Reference [17]).

Corollary 4.15. *An agent with $O((\log n)^{1-\varepsilon})$ bits of memory for an arbitrary constant $\varepsilon > 0$ needs $\Omega(\log\log n)$ pebbles to explore every graph with at most n vertices.*

5 OUTLOOK

We have established a tight bound of $\Theta(\log\log n)$ for the number of pebbles required by a single agent with constant memory for exploring any undirected graph with n vertices. We further showed that also $\Theta(\log\log n)$ agents with constant memory each are necessary and sufficient for the same task, implying that in this setting collaborating agents are not more powerful than pebbles.

An interesting question for future research is to explore to what extent these results are robust to natural variations of the model. For our algorithm, it is essential that the pebbles are *distinguishable.* A simple way to simulate p distinguishable pebbles by indistinguishable pebbles is to identify pebble $i \in \{1,\dots,p\}$ with a set of 2^i indistinguishable pebbles. However, we then need 2^p indistinguishable pebbles to simulate p distinguishable pebbles, and we would obtain an algorithm for the exploration of an undirected graph for a single agent with constant memory and $O(\log n)$ indistinguishable pebbles. While this construction shows that indistinguishable pebbles are at least as powerful as bits of memory, more work is needed to determine whether this exponential overhead for indistinguishable pebbles is necessary.

Another direction for future research would be to allow for more powerful pebbles that can, for instance, be used as bookmarks that the agent may teleport back to at any time, or more powerful agent models, e.g., with global communication between the agents, as in the jumping automaton model for graphs by Cook and Rackoff [13]. Our lower bound construction requires that communication and movements are local and cannot easily be adapted to these more powerful models.

Finally, our results built on the algorithm of Reingold [33] and, thus, concern undirected graphs only. Directed graph exploration is much harder, since an agent needs $\Omega(n\log d)$ bits of memory for exploring a directed graph with n vertices and maximum out-degree d, even when the agent has a linear number of pebbles [21]. The best-known upper bound is an agent with $O(nd\log n)$ bits and a single pebble [21], but the corresponding algorithm requires exponential time. It would be interesting to see whether the memory requirement in this setting can be reduced at the expense of a larger number of pebbles as well.

REFERENCES

[1] Susanne Albers and Monika R. Henzinger. 2000. Exploring unknown environments. *SIAM J. Comput.* 29, 4 (2000), 1164–1188.

[2] Romas Aleliunas, Richard M. Karp, Richard J. Lipton, Laszlo Lovasz, and Charles Rackoff. 1979. Random walks, universal traversal sequences, and the complexity of maze problems. In *Proceedings of the 20th Annual IEEE Symposium on Foundations of Computer Science (FOCS'79)*. 218–223.

[3] Christoph Ambühl, Leszek Gąsieniec, Andrzej Pelc, Tomasz Radzik, and Xiaohui Zhang. 2011. Tree exploration with logarithmic memory. *ACM Trans. Algor.* 7, 2 (2011), 1–21.

[4] Michael A. Bender, Antonio Fernández, Dana Ron, Amit Sahai, and Salil Vadhan. 2002. The power of a pebble: Exploring and mapping directed graphs. *Inform. Comput.* 176, 1 (2002), 1–21.

[5] Michael A. Bender and Donna K. Slonim. 1994. The power of team exploration: Two robots can learn unlabeled directed graphs. In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science (FOCS'94)*. 75–85.

[6] Avrim Blum, Prabhakar Raghavan, and Baruch Schieber. 1997. Navigating in unfamiliar geometric terrain. *SIAM J. Comput.* 26, 1 (1997), 110–137.

[7] Manuel Blum and Dexter Kozen. 1978. On the power of the compass (or, why mazes are easier to search than graphs). In *Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 132–142.

[8] Manuel Blum and William J. Sakoda. 1977. On the capability of finite automata in 2 and 3 dimensional space. In *Proceedings of the 18th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 147–161.

[9] Lothar Budach. 1978. Automata and labyrinths. *Math. Nachrichten* 86 (1978), 195–282.

[10] Jérémie Chalopin, Shantanu Das, Yann Disser, Matúš Mihalák, and Peter Widmayer. 2013. Mapping simple polygons: How robots benefit from looking back. *Algorithmica* 65, 1 (2013), 43–59.

[11] Jérémie Chalopin, Shantanu Das, Yann Disser, Matúš Mihalák, and Peter Widmayer. 2015. Mapping simple polygons: The power of telling convex from reflex. *ACM Trans. Algor.* 11, 4 (2015), 1–16.

[12] Jérémie Chalopin, Shantanu Das, and Adrian Kosowski. 2010. Constructing a map of an anonymous graph: Applications of universal sequences. In *Proceedings of the 14th International Conference on Principles of Distributed Systems (OPODIS'10)*. 119–134.

[13] Stephen A. Cook and Charles W. Rackoff. 1980. Space lower bounds for maze threadability on restricted machines. *SIAM J. Comput.* 9, 3 (1980), 636–652.

[14] Xiaotie Deng, Tiko Kameda, and Christos H. Papadimitriou. 1998. How to learn an unknown environment. I: The rectilinear case. *J. ACM* 45, 2 (1998), 215–245.

[15] Xiaotie Deng and Christos H. Papadimitriou. 1999. Exploring an unknown graph. *J. Graph Theory* 32, 3 (1999), 265–297.

[16] Krzysztof Diks, Pierre Fraigniaud, Evangelos Kranakis, and Andrzej Pelc. 2004. Tree exploration with little memory. *J. Algor.* 51, 1 (2004), 38–63. DOI: https://doi.org/10.1016/j.jalgor.2003.10.002

[17] Yann Disser, Jan Hackfeld, and Max Klimm. 2015. Undirected graph exploration with $\Theta(\log \log n)$ pebbles. In *Proceedinbgs of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'15)*. 25–39.

[18] Gregory Dudek, Michael Jenkin, Evangelos E. Milios, and David Wilkes. 1991. Robotic exploration as graph construction. *IEEE Trans. Robot. Autom.* 7, 6 (1991), 859–865.

[19] Rudolf Fleischer and Gerhard Trippen. 2005. Exploring an unknown graph efficiently. In *Proceedings of the 13th Annual European Symposium on Algorithms (ESA'05) (LNCS)*, G. Brodal and S. Leonardi (Eds.), Vol. 3669. 11–22.

[20] Klaus-Tycho Foerster and Roger Wattenhofer. 2016. Lower and upper competitive bounds for online directed graph exploration. *Theoret. Comput. Sci.* 655 (2016), 15–29.

[21] Pierre Fraigniaud and David Ilcinkas. 2004. Digraphs exploration with little memory. In *Proceedings of the 21st Annual Symposium on Theoretical Aspects of Computer Science (STACS'04)*. 246–257.

[22] Pierre Fraigniaud, David Ilcinkas, Guy Peer, Andrzej Pelc, and David Peleg. 2005. Graph exploration by a finite automaton. *Theoret. Comput. Sci.* 345, 2–3 (2005), 331–344.

[23] Pierre Fraigniaud, David Ilcinkas, and Andrzej Pelc. 2008. Impact of memory size on graph exploration capability. *Discrete Appl. Math.* 156, 12 (2008), 2310–2319.

[24] Pierre Fraigniaud, David Ilcinkas, Sergio Rajsbaum, and Sébastien Tixeuil. 2006. The reduced automata technique for graph exploration space lower bounds. In *Theoretical Computer Science*, O. Goldreich, A. L. Rosenberg, and A. L. Selman. Lecture Notes in Computer Science, Vol. 3895. Springer, Berlin, Heidelberg.

[25] Frank Hoffmann. 1981. One pebble does not suffice to search plane labyrinths. In *Proceedings of the 3rd International Symposium on Fundamentals of Computation Theory (FCT'81)*. 433–444.

[26] Sorin Istrail. 1988. Polynomial universal traversing sequences for cycles are constructible. In *Proceedings of the 20th Annual ACM Symposium on Theory Computing (STOC'88)*. 491–503.

[27] Bala Kalyanasundaram and Kirk Pruhs. 1994. Constructing competitive tours from local information. *Theoret. Comput. Sci.* 130, 1 (1994), 125–138.
[28] Michal Koucký. 2002. Universal traversal sequences with backtracking. *J. Comput. Syst. Sci.* 65, 4 (2002), 717–726.
[29] Michal Koucký. 2003. *On Traversal Sequences, Exploration Sequences and Completeness of Kolmogorov Random Strings.* Ph.D. Dissertation. Rutgers, The State University of New Jersey.
[30] Nicole Megow, Kurt Mehlhorn, and Pascal Schweitzer. 2012. Online graph exploration: New results on old and new algorithms. *Theoret. Comput. Sci.* 463 (2012), 62–72.
[31] Shuichi Miyazaki, Naoyuki Morimoto, and Yasuo Okabe. 2009. The online graph exploration problem on restricted graphs. *IEICE Trans. Info. Syst.* E92-D, 9 (2009), 1620–1627.
[32] Noam Nisan. 1992. Pseudorandom generators for space-bounded computation. *Combinatorica* 12, 4 (1992), 449–461.
[33] Omer Reingold. 2008. Undirected connectivity in log-space. *J. ACM* 55, 4 (2008), 17.
[34] Hans-Anton Rollik. 1980. Automaten in planaren Graphen. *Acta Inform.* 13 (1980), 287–298.
[35] Daniel J. Rosenkrantz, Richard E. Stearns, and Philip M. Lewis, II. 1977. An analysis of several heuristics for the traveling salesman problem. *SIAM J. Comput.* 6, 3 (1977), 563–581.
[36] Walter J. Savitch. 1973. Maze recognizing automata and nondeterministic tape complexity. *J. Comput. Syst. Sci.* 7 (1973), 389–403.
[37] Anupam N. Shah. 1974. Pebble automata on arrays. *Comput. Vision Graph.* 3, 3 (1974), 236–246.
[38] Claude A. Shannon. 1951. Presentation of a maze-solving machine. In *Proceedings of the 8th Conference on Cybernetics.* 173–180.

Received April 2018; revised July 2019; accepted July 2019