Undirected Graph Exploration with $\Theta(\log\log n)$ Pebbles[*]

Yann Disser[†] Jan Hackfeld[†] Max Klimm[†]

Abstract

We consider the fundamental problem of exploring an undirected and initially unknown graph by an agent with little memory. The vertices of the graph are unlabeled, and the edges incident to a vertex have locally distinct labels. In this setting, it is known that $\Theta(\log n)$ bits of memory are necessary and sufficient to explore any graph with at most n vertices. We show that this memory requirement can be decreased significantly by making a part of the memory distributable in the form of pebbles. A pebble is a device that can be dropped to mark a vertex and can be collected when the agent returns to the vertex. We show that for an agent $\mathcal{O}(\log\log n)$ distinguishable pebbles and bits of memory are sufficient to explore any bounded-degree graph with at most n vertices. We match this result with a lower bound exhibiting that for any agent with sub-logarithmic memory, $\Omega(\log\log n)$ distinguishable pebbles are necessary for exploration.

1 Introduction

The exploration of large graphs is a central challenge in many applications ranging from robot navigation in unknown environments [1, 4] to Web crawling [23] and image recognition [6, 8, 27]. In this paper, we are interested in the problem of deterministically visiting all vertices of an initially unknown undirected graph by a single agent. To account for the fact that, in many interesting scenarios, the size of the graph exceeds the agent's internal memory by orders of magnitude, we study the exploration problem from the viewpoint of space complexity. It is known that an agent needs $\Theta(\log n)$ bits of memory to explore any graph with at most n vertices. In fact, already the subproblem of deciding whether two given vertices s and t are connected in an undirected graph is complete for SL, the class of problems solvable by symmetric non-deterministic log-space computations [24].

To break this logarithmic barrier, we allow for distributable memory in the form of pebbles. Pebbles are distinguishable devices that can be dropped to mark a vertex and that can be retrieved when revisiting it. Since the memory required to store a unique vertex identifier exceeds the internal memory of the agent, it is reasonable to assume that the agent has no direct means of distinguishing vertices, other than pebbles. To allow sensible navigation of the agent in the absence of vertex identifiers, we assume that the edges incident to each vertex have locally unique labels.

Our results and significance. We show that for any agent with sub-logarithmic memory, $\Theta(\log\log n)$ distinguishable pebbles are both necessary and sufficient to explore any graph with n vertices. In fact, already $\mathcal{O}(\log\log n)$ bits of memory and $\mathcal{O}(\log\log n)$ pebbles are sufficient for the exploration.

Our proof of this upper bound is constructive, i.e., we devise an algorithm that explores any graph with n vertices using $\mathcal{O}(\log\log n)$ pebbles and $\mathcal{O}(\log\log n)$ memory. Our algorithm does not require n to be known, terminates after $n^{\mathcal{O}(\log\log n)}$ steps and returns to the starting location. In terms of the s-t-connectivity problem, this implies that the space complexity of $\Theta(\log n)$ [15, 24] can be significantly reduced if we allow memory to be distributable in the form of pebbles.

To prove the lower bound, we construct a family of graphs with $\mathcal{O}(s^{8^{p+1}})$ vertices that trap any agent with s states and p pebbles. The previously best bounds are for a more general setting where each pebble is an autonomous agent with constant memory. The traps of Rollik [26] are of order $\tilde{\mathcal{O}}(s \uparrow\uparrow (2p+1))$, where $a \uparrow\uparrow b := a^{a^{\cdot^{\cdot^{a}}}}$ with b levels in the exponent. This bound was improved by Fraigniaud et al. [16] to $\tilde{\mathcal{O}}(s \uparrow\uparrow (p+1))$. Our construction exhibits that dramatically smaller traps with only a doubly exponential number of vertices are possible when pebbles cannot move autonomously. As a consequence of our trap, we are able to show that, for any constant $\epsilon > 0$, any agent with $\mathcal{O}((\log n)^{1-\epsilon})$ bits of memory needs at least $\Omega(\log\log n)$ distinguishable pebbles for the exploration task.

Further related work. A cornerstone of the graph exploration literature is the work of Aleliunas et al. [2], who showed that a random walk in an undirected graph visits all vertices in a polynomial number of steps. This insight gives rise to a randomized constant-space perpetual exploration algorithm, i.e., an algorithm that runs forever and, eventually, visits all vertices of the

[*]This research was supported by the DFG Priority Programme 1736 "Algorithms for Big Data".

[†]TU Berlin, Institute of Mathematics, MA 5-2, Str. des 17. Juni 136, 10623 Berlin, Germany

graph. If an upper bound on the number of vertices of the graph is known, the random walk can be turned into a randomized log-space exploration algorithm that terminates after having explored the graph (with high probability) by adding a counter that keeps track of the numbers of steps taken.

In this context, deterministic exploration algorithms, as considered in this work, can be seen as ways to derandomize random walks. A famous result in this direction is due to Reingold [24] who gave a log-space deterministic exploration algorithm for undirected graphs. This algorithm is perpetual, but can be modified to terminate if an upper bound on the number of vertices is known. The space complexity of Reingold's algorithm is best possible, since perpetual exploration of undirected graphs requires $\Omega(\log n)$ space, as shown by Fraigniaud et al. [15].

The results above can be also be phrased in terms of traversal and exploration sequences. These are sequences of local edge labels that govern the walk of the agent in the graph. A sequence is universal for a class of graphs, if any graph in the class is explored by it. Aleliunas et al. showed that random sequences of length $\mathcal{O}(\Delta^2 n^3 \log n)$ are universal traversal sequences for all undirected Δ-regular graphs with high probability. The explicit construction of universal traversal sequences of polynomial length seems to be more challenging and only for restrictive graph classes, such as cycles and expander graphs, such constructions were given [18, 19, 21]. Koucky [20] introduced exploration sequences that, in contrast to traversal sequences, allow for backtracking, and showed that this framework facilitates explicit constructions of sequences of polynomial length for special classes of graphs. Reingold's algorithm constructs a universal exploration sequence for all undirected graphs in logarithmic space and, hence, of polynomial length. In subsequent work, Reingold et al. [25] extended this construction and obtained log-space constructable universal traversal sequences for all regular directed graphs with a restricted type of labeling.

For trees with maximum degree Δ, Diks et al. [12] gave a perpetual exploration algorithm that uses $\mathcal{O}(\log \Delta)$ space, i.e., asymptotically not more than the space needed to store a single edge label. They showed that $\Omega(\log \log \log n)$ bits of memory are needed if the algorithm has to eventually terminate. If, in addition, the algorithm is required to terminate at the same vertex where it started, $\Omega(\log n)$ bits of memory are needed. A matching upper bound for the latter was given by Ambuhl et al. [3].

Regarding the exploration time, Dudek et al. [13] showed that an agent provided with a pebble can map an undirected graph in time $\mathcal{O}(n^2\Delta)$. In a similar vein, Chalopin et al. [11] showed that if the starting node can be recognised by the agent, the agent can explore the graph in time $\mathcal{O}(n^3\Delta)$ using $\mathcal{O}(n\Delta \log n)$ memory.

Exploration becomes considerably more difficult for directed graphs, where random walks may need exponential time to visit all vertices. Without any constraints on memory, Bender et al. [5] gave an $\mathcal{O}(n^8\Delta^2)$-time algorithm that uses one pebble and explores (and maps) a 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 [14] 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 $\mathcal{O}(n\Delta \log n)$-space algorithm for terminating exploration with an exponential running time using a single pebble. They also gave an $\mathcal{O}(n^2\Delta \log n)$-space algorithm running in polynomial time and using $\mathcal{O}(\log \log n)$ indistinguishable pebbles. According to Bender et al. [5] at least $\Omega(\log \log n)$ pebbles are necessary in this setting.

Further related is the problem of exploring geometric structures in the plane, see, e.g., [7, 17, 27] for the exploration of labyrinths, and [9, 10] for the the exploration of simple polygons.

2 Terminology and model

We consider an agent that is initially located at some vertex of a connected, undirected graph $G = (V, E)$ without any knowledge of the overall graph topology. The goal of the agent is to *explore* the graph, i.e., to systematically visit all of its vertices. We assume that the graph is anonymous, i.e., the vertices of the graph are unlabeled and therefore cannot be identified by the agent. However, for the agent to be able to navigate locally, we assume that the edges incident to a vertex v have distinct labels $0, \dots, d_v - 1$, where d_v is the degree of v. This way every edge $\{u, v\}$ has two labels called *port numbers*, one at u and one at v. In each step, the agent observes the degree of the current vertex and the port number of the edge by which it entered the vertex. It can then select a port number of any of the incident edges and traverse the corresponding edge to the other endpoint. Note that we assume no relation between the two port numbers of an edge in this model.

We further extend the model by equipping the agent with a set of p distinguishable pebbles, which can be thought of as bits of memory that can be distributed as markers or bookmarks. The agent can carry the pebbles, place any of the pebbles on a vertex, observe which pebbles are placed on the current location, and it can pick up pebbles encountered at its location. Thus, the pebbles give the agent the ability to distinguish and recognize vertices and therefore provide additional information for choosing the next edges to traverse. We refer to the two types of models for the agent as an *agent without pebbles* or an *agent with pebbles*, respectively. Generally, we are interested in exploring graphs with as few pebbles as possible.

Formally, we model the agent as a Turing machine with a finite tape that determines the traversal of the agent in the graph. We call this model a (s,p,m)-*pebble machine*, where s is the number of states, p is the number of pebbles and m is the tape length of the Turing machine. In every step, the agent observes the local environment at the current vertex in the graph G according to a function δ_{in}, then it does an arbitrary number of computation steps on its tape according to a transition function δ until it reaches a final state, and afterwards it performs actions according to a function δ_{out}. More precisely, a (s,p,m)-pebble machine $\mathcal{M}$ is a tuple $(Q, F, P, m, \delta, \delta_{\text{in}}, \delta_{\text{out}}, q_0)$, where Q is the set of states of $\mathcal{M}$, q_0 is the starting state, $F \subseteq Q$ the set of final states and P is the set of pebbles. The pebble machine has a tape of length m with tape alphabet $\{0,1\}$ and does computations according to a function

$$\delta\colon (Q \setminus F) \times \{0,1\} \to Q \times \{0,1\} \times \{L,R\},$$

which is the same as for a regular Turing machine. That is, if the pebble machine is in state q, reads symbol $a \in \{0,1\}$ on the tape and $\delta(q,a) = (q', a', d)$, then it changes to state q', writes a' to the tape and moves left if $d = L$ and right if $d = R$. For all pebble machines that we consider, we assume that the computation terminates in a final state $q \in F$ after a finite number of transitions according to δ.

The pebble machine $\mathcal{M}$ observes the local environment at the current vertex of an undirected graph G with maximum degree Δ according to the function

$$\delta_{\text{in}}\colon Q \times 2^P \times 2^P \times \{0,\dots,\Delta-1\}^2 \to Q,$$

in the following way. If $\delta_{\text{in}}(q, P_1, P_2, d, l) = q'$, then $\mathcal{M}$ transitions to state q', provided that the current state is q, the agent carries the set of pebbles P_1, observes the set of pebbles P_2 at the current location, that the degree of the current vertex is d and that it has entered the current vertex via the edge labeled l. Initially, $\mathcal{M}$ is in some vertex v_0 in G in state q_0, observes the degree of v_0 and we assume $l = 0$, $P_1 = P$, $P_2 = \emptyset$. Note that δ_{in} is a partial function, as it is only defined for $P_1 \cap P_2 = \emptyset$, since a pebble cannot both be carried by the agent and placed on the current vertex.

The actions performed by $\mathcal{M}$ are determined by the function

$$\delta_{\text{out}}\colon Q \to 2^P \times 2^P \times \{0,\dots,\Delta-1\}.$$

If $\delta_{\text{out}}(q) = (P_1, P_2, l)$ and the agent is in state q, it performs the following sequence of actions. It places all pebbles that are carried and contained in P_1 on the current vertex, it picks up all pebbles that are placed on the current vertex and contained in P_2, and after that traverses the edge labeled l. Naturally, we demand the overall set of pebbles carried or placed on a vertex to remain unchanged before and after executing the actions according to δ_{out}, i.e., $P_1 \cup P_2 = P_1' \cup P_2'$ with $P_1' \cap P_2' = \emptyset$ if $\delta_{\text{in}}(q, P_1, P_2, d, l) = q'$ and $\delta_{\text{out}}(q') = (P_1', P_2', l')$.

A *configuration* of a pebble machine is determined by the current state of the pebble machine, the current head position and the content of the tape. Thus, a (s,p,m)-pebble machine has at most $sm2^m$ possible configurations.

Note that we model the agent as a pebble machine instead of an automaton as we need a certain "locality" of the transitions. An automaton can change from any state to an arbitrary other state, whereas the tape of the pebble machine only changes at the current head position. This allows us in the next section to simulate the tape of a pebble machine by pebbles placed on vertices in the graph and to simulate the computations of the pebble machine by moving these pebbles.

3 An exploration algorithm using pebbles

The main result in this section is an algorithm that explores any bounded-degree graph on at most n vertices with $\mathcal{O}(\log\log n)$ pebbles and memory.

For our algorithm, we use the concept of *exploration sequences* introduced by Koucký [20]. An exploration sequence is a sequence of integers $e_0, e_1, e_2, \dots$ that guides the walk of an agent through a graph G as follows. Assume the agent starts in a vertex v_0 of G and let $l_0 = 0$. Let v_i denote the agent's location in step i and l_i the label of the edge at v_i leading back to the previous location. Then, the agent *follows* the exploration sequence 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. 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. If an exploration sequence becomes periodic at some point, we write it as $e_0, \dots, e_{i-1}(e_i \dots, e_j)^*$ for suitable $i, j \in \mathbb{N}$ with $i < j$. This notation means that the part $e_i \dots, e_j$ is repeated infinitely often after e_{i-1}.

The following result of Reingold is one of the main building blocks of our exploration algorithm.

Theorem 3.1. ([24], Corollary 5.5) *There is an $\mathcal{O}(\log n)$-space algorithm producing a universal exploration sequence for any regular graph on n vertices.*

The general idea of our exploration algorithm is as follows. We use a constant amount of memory to execute a modified version of Reingold's algorithm that brings the agent back to the starting vertex. The resulting closed walk ω is guaranteed to contain a number of distinct vertices n' that is exponential in the memory used. Now, we simulate additional memory (of larger size than our original memory) by using a constant number of our pebbles: The position of each pebble along the closed walk ω encodes one digit of our memory in base n'. A technical challenge is to show how to manage the transition between memory states and, in particular, how to deal with vertices that appear multiple times along ω.

In order to use the additional memory for an exploration algorithm, we need to be able to operate on ω after each step of the exploration. This means that the agent needs a way of "carrying ω along" after each move. We show that it is possible to find a new closed walk ω' after each step and move the pebbles one by one from ω to ω' while preserving the content of the memory.

Finally, we use the additional memory to execute the modified version of Reingold's algorithm again, this time yielding a walk with $n'' > n'$ distinct vertices. We again use this new walk as memory and continue recursively, until we get a walk with n distinct vertices. At this point, the graph is explored. Note that pebbles are distinguishable and we need not worry about confusing pebbles used in different levels of the recursion.

We show how to implement this general procedure with $\mathcal{O}(\log\log n)$ pebbles and $\mathcal{O}(\log\log n)$ bits of memory. As a first step, we show in Theorem 3.2 how to modify Reingold's algorithm to yield a closed walk containing an exponential number of vertices in terms of the memory used. Afterwards, in the proof of Theorem 3.3, we explain how to place pebbles on the closed walk and use them as additional memory.

Theorem 3.2. *There exists a $(\mathcal{O}(1), 0, \mathcal{O}(\log n))$-pebble machine that moves along a closed walk and either explores the graph or visits at least n distinct vertices, for any graph with bounded degree.*

For proving Theorem 3.2, we first establish that following a periodic exploration sequence yields a closed walk.

Lemma 3.1. *An agent following an exploration sequence of the form $(e_0, \dots, e_{k-1})^*$ in an undirected graph moves along a closed walk.*

Proof. Let $\mathcal{A}$ be an agent following an exploration sequence $(e_0, \dots, e_{k-1})^*$ starting at a vertex v_0 of some graph G. The walk of $\mathcal{A}$ in G translates to a walk in a directed graph G' which is defined as follows: The vertices in G' are tuples (v, l, i), where $v \in V(G)$, l is the port number at v of the edge by which the agent entered v and $i \in \{0, \dots, k-1\}$ is the index of the next element in the exploration sequence. Moreover, there is an edge from (v, l, i) to (v', l', i') in G' if $i' = i + 1 \mod k$, there is an edge $\{v, v'\}$ in G with port number l' at v' and $l + e_i \mod d_v$ at v, where d_v is the degree of v. Note that if $v_0, v_1, \dots$ is the sequence of vertices visited by $\mathcal{A}$ and every vertex v_j is entered via the edge with port number l_j, then $(v_0, l_0, 0), (v_1, l_1, 1 \mod k), (v_2, l_2, 2 \mod k), \dots$ is a walk in the graph G'.

We show that any vertex (v, l, i) in G' has exactly one outgoing edge and at most one incoming edge. The first is obvious. For the second, assume that there is an edge from both (v', l', i') and (v'', l'', i'') to (v, l, i). This means that both from v' and v'' the agent entered v via the edge with port number l at v implying $v' = v''$. Moreover, $i' = i'' = i - 1 \mod k$ holds, and we have $l' + e_{i'} \mod d_{v'} = l'' + e_{i''} \mod d_{v''}$, where $d_{v'}$ is the degree of v'. Thus $l' = l''$ holds, since the labels are both in $\{0, \dots, d_{v'}\}$.

As each vertex has exactly one outgoing edge and at most one incoming edge, following the unique walk starting from $(v_0, l_0, 0)$ must eventually lead back to this vertex because G' is finite. Thus, $(v_0, l_0, 0)$ lies on a circle in G' and the projection of the walk on this circle onto the first component yields the walk of $\mathcal{A}$ in G. This implies that the starting vertex v_0 is visited infinitely often. □

We now adapt the algorithm from Theorem 3.1 to work for 3-regular graphs with up to (rather than exactly) n vertices and to produce a universal exploration sequence of the form $(e_0, \dots, e_{k-1})^*$, i.e., an exploration sequence yielding a closed walk.

Lemma 3.2. *There exists an $\mathcal{O}(\log n)$-space algorithm producing a universal exploration sequence $(e_0, \dots, e_{k-1})^*$ for any 3-regular graph with at most n vertices.*

Proof. Note that all 3-regular graphs have an even number of vertices and it therefore suffices to show the claim for even n. Let $\mathcal{M}$ be the Turing machine of Theorem 3.1, producing a universal exploration sequence for any 3-regular graph on exactly n vertices. We first show that the exploration sequence is actually universal for any 3-regular graph on at most n vertices and then modify $\mathcal{M}$ so that is produces an exploration sequence of the form $(e_0, \ldots, e_{k-1})^*$.

For the sake of contradiction, let G_0 be a 3-regular graph with the maximum number of vertices $n_0 \leq n$ such that an agent $\mathcal{A}$ following the exploration sequence produced by $\mathcal{M}$ does not explore for some starting vertex v_0. Then $n_0 < n$ by the choice of $\mathcal{M}$ and n_0 is even because G_0 is 3-regular. Let v be a vertex not visited by $\mathcal{A}$ and w_1, w_2, w_3 be its neighbors. We replace v by a clique of three vertices v_1, v_2, v_3 and add the edges $\{v_i, w_i\}$ for $i = 1, 2, 3$ to G_0 with suitable labels. We obtain a 3-regular graph G_1 with $n_1 := n_0 + 2$ vertices. As the remaining graph is unchanged, G_1 is also not explored by $\mathcal{A}$ when starting in v_0. This is a contradiction to the maximality of n_0.

We want to modify $\mathcal{M}$ so that it produces an exploration sequence of the form $(e_0, \ldots, e_{k-1})^*$, which is still universal for any 3-regular graph on at most n vertices. Let c be the number of configurations of $\mathcal{M}$ and $k := 3n \cdot c$. Further, let $\mathcal{M}'$ be the Turing machine that runs $\mathcal{M}$ for k steps and then restarts. Note that $\mathcal{M}'$ can be realized by adding an additional counter to $\mathcal{M}$ that counts up to k. By construction, $\mathcal{M}'$ produces an exploration sequence $(e_0, \ldots, e_{k-1})^*$. Moreover, $\mathcal{M}'$ still needs $\mathcal{O}(\log n)$ space as counting up to k can be done in $\mathcal{O}(\log n)$ space.

What is left to show is that $\mathcal{M}'$ produces a universal exploration sequence for any 3-regular graph on at most n vertices. For the sake of contradiction, assume there exists some 3-regular graph G on at most n vertices so that an agent $\mathcal{A}'$ starting in a vertex v_0 and following the exploration sequence produced by $\mathcal{M}'$ does not explore G. Then $\mathcal{A}'$ only explores a strict subset V' of the vertices V of G. The first k numbers in the exploration sequence of $\mathcal{M}$ and $\mathcal{M}'$ are the same and thus an agent $\mathcal{A}$ following the exploration sequence produced by $\mathcal{M}$ also only visits vertices in V' in the first k steps. In these first k steps $\mathcal{A}$ visits $|V'| < n$ vertices, each of which it can enter via one of 3 possible edges. The next edge is uniquely determined by one of c possible configurations of the Turing machine $\mathcal{M}$. Therefore, in the first k steps $\mathcal{A}$ will twice enter the same vertex via the same edge with $\mathcal{M}$ being in the same configuration. This means that $\mathcal{A}$ will continue in a closed walk visiting only vertices in V'. This contradicts the universality of the exploration sequence produced by $\mathcal{M}$. □

We are now ready to give the proof of Theorem 3.2.

Proof. Given a graph G with the maximum degree bounded by a constant Δ, we use the following construction taken from [22, Theorem 87] to transform G to a

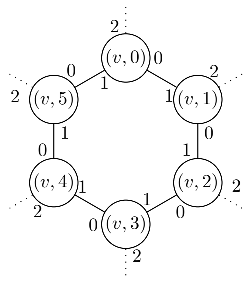

Figure 1: Example for the transformation of a graph G to a 3-regular graph G_{reg}. A vertex v of degree 2 is transformed to the following circle containing 6 vertices.

3-regular graph G_{reg}. 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 $\{(v, i), (v, i + 1 \mod 3d_v)\}$ has port number 0 at (v, i) and 1 at $(v, i + 1 \mod 3d_v)$. An example is given in Figure 1. 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}. The idea now is to apply Lemma 3.2 for the 3-regular graph G_{reg} and transform this walk back to the original graph G.

By Lemma 3.2, there exists a Turing machine with $\mathcal{O}(1)$ states using a tape of length $\mathcal{O}(\log n)$ and producing an exploration sequence $(e_0, e_1, \ldots, e_{k-1})^*$ for any 3-regular graph on at most $3\Delta n^2$ vertices. We construct a pebble machine $\mathcal{M}$ that internally runs this algorithm and transforms the simulated walk on G_{reg} to the corresponding traversal of G. The simulated walk of $\mathcal{M}$ in G_{reg} starts in $(v_0, 0)$ and the walk in G is the projection of the simulated walk onto the first component. Note that $\mathcal{M}$ only needs to store the second component of the current vertex (v, i) in G_{reg}. The first component remains unchanged when choosing an edge with label 0 or 1 in G_{reg}. Furthermore, when choosing an edge with label 2 in G_{reg}, then $\mathcal{M}$ has to take the edge labeled $i \mod d_v$ at v to a distinct vertex w. If j is the label of the edge at w, then (w, j) is the current vertex in G_{reg} and it suffices to store j. Overall, $\mathcal{O}(1)$ states are sufficient for reading the degree of the current vertex and port number of the last edge according to δ_{in} and traversing edges according to δ_{out}, because the maximum degree is bounded by the constant Δ. Thus, $\mathcal{M}$ can be implemented as a $(\mathcal{O}(1), 0, \mathcal{O}(\log n))$-pebble machine.

Furthermore, by Lemma 3.1, $\mathcal{M}$ visits $(v_0, 0)$ infinitely often along the simulated walk on G_{reg} and therefore it visits v_0 infinitely often in the walk on G. This shows that $\mathcal{M}$ does a closed walk.

What is left to show is that $\mathcal{M}$ explores G or visits at least n vertices. Assume neither is the case. Let G' be the induced subgraph of G containing all vertices visited by $\mathcal{M}$ and all neighbors of these vertices. Then G' contains at most $n + (\Delta - 1) \cdot n \leq n^2$ vertices. Moreover, the walk of $\mathcal{M}$ on G' starting in v_0 exactly corresponds to the walk of $\mathcal{M}$ on G. In particular, $\mathcal{M}$ does not explore G'. Let G'_{reg} be the transformation of G' to a 3-regular graph according to [22, Theorem 87]. Then G'_{reg} contains at most $3\Delta n^2$ vertices. However, the walk of $\mathcal{M}$ on G' corresponds to the walk on G'_{reg} given by a universal exploration sequence $(e_0, e_1, \ldots, e_{k-1})^*$ for any 3-regular graph of on at most $3\Delta n^2$ vertices. But this means $\mathcal{M}$ is guaranteed to explore G', a contradiction. $\square$

Using Theorem 3.2, we show how to boost our memory using a subset of our pebbles.

Theorem 3.3. *There is a constant $c \in \mathbb{N}$, such that for any graph G with bounded degree and any $(s, p, 2m)$-pebble machine $\mathcal{M}$, there exists a $(cs, p + c, m)$-pebble machine $\mathcal{M}'$ that simulates the walk of $\mathcal{M}$ or explores G.*

Proof. Let Q be the set of states of $\mathcal{M}$. We define the set of states of $\mathcal{M}'$ to be $Q \times Q'$ for a set Q', i.e., any state of $\mathcal{M}'$ is a tuple (q, q'), where q corresponds to the state of $\mathcal{M}$ in the current step of the traversal. The pebble machine $\mathcal{M}'$ observes the input according to δ_{in} and performs actions according to δ_{out} just as $\mathcal{M}$, while only changing the first component of the current state. $\mathcal{M}'$ uses p pebbles in the same way as $\mathcal{M}$ and possesses a set $\{p_{\text{start}}, p_{\text{temp}}, p_{\text{next}}, p_0, p_1, \ldots, p_{c-4}\}$ of additional pebbles. The pebble p_{start} is dropped by $\mathcal{M}'$ right after observing the input according to δ_{in} in order to mark the current location of $\mathcal{M}$ during the traversal. The purpose of the pebbles p_{temp} and p_{next} will be explained later. The other pebbles $\{p_0, p_1, \ldots, p_{c-4}\}$ are placed along a closed walk ω to simulate the memory of $\mathcal{M}$, while the states Q' and the tape of $\mathcal{M}'$ are used to manage this memory.

To this end, we divide the tape of $\mathcal{M}'$ 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 $\mathcal{M}$ 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 $\mathcal{M}'$. By Theorem 3.2, there is a constant c_1 and a $(c_1, 0, c_1 k)$-pebble machine that moves along a closed walk and visits at least 2^k distinct vertices in any graph of containing at least 2^k vertices. Let $m_1 := \lfloor m/(c_0 c_1) \rfloor$ 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 $2^{m'/c_0} > 2m'$.

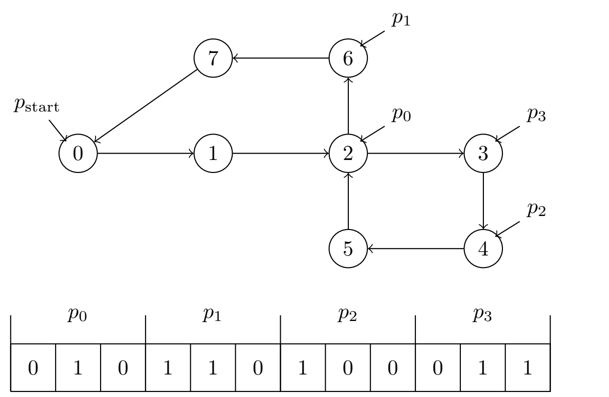

Figure 2: Example of a closed walk, where $c - 3 = 4$ pebbles are used to encode the configuration of a tape of length $2m = 12$. Directed edges indicate the order of the vertices along ω. The position of each pebble encodes $m_1 = 3$ bits.

In the following, we show how the simulated memory is managed by providing algorithms in pseudocode (see algorithms 1 to 4). These can be implemented on a Turing machine with a constant number of states c_{Alg}. Let $c = \max\{2^{2m_0}, 2c_0c_1 + 3, c_{\text{Alg}}\}$. 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 $\mathcal{M}$ in the states Q' of $\mathcal{M}'$, since $c \geq 2^{2m_0}$.

We proceed to show that the computations on the tape of length $2m$ performed by $\mathcal{M}$ according to the transition function δ can be simulated using the pebbles $\{p_{\text{start}}, p_{\text{temp}}, p_0, p_1, \ldots, p_{c-4}\}$. The proof of this result proceeds along the following key claims.

1. We can find a closed walk ω containing 2^{m_1} distinct vertices so that $c - 3$ pebbles placed along this walk can encode all configurations of the tape of $\mathcal{M}$.

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 the closed walk ω starts at vertex v and $\mathcal{M}$ moves from vertex v to vertex v', 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. Theorem 3.2 yields a $(c_1, 0, m/c_0)$-pebble machine $\mathcal{M}_{\text{walk}}$ moving along a closed walk ω. We use a variable T_{walk} of size m/c_0

Algorithms 1 Auxiliary functions for moving along the closed walk ω.

```
function Step()
    traverse edge according to $\mathcal{M}_{\text{walk}}$
    $T_{\text{steps}} \leftarrow T_{\text{steps}} + 1$

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

function Restart()
    FindPebble($p_{\text{start}}$)
    $T_{\text{steps}} \leftarrow 0$
    $T_{\text{id}} \leftarrow 0$
    $T_{\text{walk}} \leftarrow 0$
```

Algorithm 2 Moving along the closed walk ω while updating T_{steps} and T_{id}.

```
function NextDistinctVertex()
    if $T_{\text{id}} \equiv 2^{m_1} - 1$
        Restart()
        return
    $T_{\text{id}} \leftarrow T_{\text{id}} + 1$
    $T'_{\text{steps}} \leftarrow T_{\text{steps}}$
    repeat
        Step()
        $T'_{\text{steps}} \leftarrow T'_{\text{steps}} + 1$
        drop($p_{\text{temp}}$)
        $T'_{\text{walk}} \leftarrow T_{\text{walk}}$
        Restart()
        FindPebble($p_{\text{temp}}$)
        pickup($p_{\text{temp}}$)
        $T_{\text{walk}} \leftarrow T'_{\text{walk}}$
    until $T_{\text{steps}} = T'_{\text{steps}}$
```

for the memory of $\mathcal{M}_{\text{walk}}$, which is initially assumed to have all bits set to 0. If $\mathcal{M}_{\text{walk}}$ explores G, we are done. Otherwise, ω must contain at least 2^{m_1} distinct vertices. We need to show that $c-3$ pebbles placed along the walk ω can be used to encode any of the 2^{2m} configurations of the tape of $\mathcal{M}$. Figure 2 shows how each pebble encodes a certain part of the tape of $\mathcal{M}$. 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 $\mathcal{M}$ to be numbered $0, 1 \ldots, 2m-1$. As $c \geq 2c_0c_1 + 3$, we have enough pebbles to encode the configuration of the tape of $\mathcal{M}$.

2. Navigating ω. Let T_{steps} be a variable counting the number of steps along ω and T_{id} be a variable for counting the number of unique vertices visited along ω after starting in the vertex marked by p_{start}. Note that T_{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 number of steps along the closed walk may be arbitrary large, as we do not impose any restrictions on the number of vertices of G and vertices may be visited multiple times. For our purposes it suffices to ensure that T_{steps} contains the correct count, as long as we have not seen more than 2^{m_1} distinct vertices. Let g be the number of configurations of $\mathcal{M}_{\text{walk}}$. Then $g \leq c_1 \cdot m/c_0 \cdot 2^{m/c_0}$ because there are c_1 possible states, m/c_0 possible head positions and $2^{m/c_0}$ possible configurations of the tape. After $g2^{m_1}$ steps we must have visited at least 2^{m_1} vertices, as otherwise we would be in the same vertex in the same configuration twice and therefore loop while visiting less than 2^{m_1} vertices. It is thus sufficient to count up to $g2^{m_1}$. We have

$$g2^{m_1} \leq c_1 \cdot m/c_0 \cdot 2^{m/c_0} \cdot 2^{m_1} \leq \left(2^{m/c_0}\right)^4,$$

where we used that $m \geq m_0$ and therefore $c_1 \leq 2^{m/c_0}$. Thus, $4m/c_0$ tape cells are sufficient for the variable T_{steps} to count up to $g2^{m_1}$.

It remains to show that we can move along the closed walk ω while updating T_{steps} and T_{id}, such that, starting from the vertex marked by p_{start}, the variable T_{steps} contains the number of steps taken and T_{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 if possible, and let observe(p_i) be "true" if pebble p_i is located at the current position. Consider Algorithms 1. The function Step() moves one step along ω and updates T_{steps} accordingly. The function FindPebble(p_i) moves along ω until it finds pebble p_i. The function Restart() goes back to the starting vertex marked by p_{start}, sets both variables T_{steps} and T_{id} to 0, and restarts $\mathcal{M}_{\text{walk}}$ by setting the variable T_{walk} to 0. Finally, the function NextDistinctVertex() in Algorithm 2 does the following: If the number of distinct vertices visited is already 2^{m_1}, then we go back to the start. Otherwise, we continue along ω until we encounter a vertex we have not visited before. We repeatedly traverse an edge, drop the pebble p_{temp}, store the number of steps until reaching that vertex, then we restart from the beginning and check if we can reach that vertex with fewer steps. If not, we found a new distinct vertex. Note that we use the auxiliary variables T'_{steps} and T'_{walk}, which both need a constant number of blocks of size m_0/c_0.

3. Reading from and writing to simulated memory. We show how we can simulate the changes to the

Algorithms 3 Reading and changing positions of pebbles.

```
function GetPebbleId($p_i$)
    Restart()
    while not observe($p_i$) do
        NextDistinctVertex()
    return $T_{\mathrm{id}}$
function PutPebbleAtId($p_i$,id)
    FindPebble($p_i$)
    pickup($p_i$)
    Restart()
    while id>0 do
        id $\leftarrow$ id $- 1$
        NextDistinctVertex()
    drop($p_i$)
```

Algorithms 4 Reading and writing one bit for the simulated memory.

```
function ReadBit()
    $i \leftarrow \lfloor T_{\mathrm{head}}/m_1 \rfloor$
    $j \leftarrow T_{\mathrm{head}} - m_1 \cdot i$
    id $\leftarrow$ GetPebbleId($p_i$)
    return bit $j$ of id
function WriteBit($b$)
    $i \leftarrow \lfloor T_{\mathrm{head}}/m_1 \rfloor$
    $j \leftarrow T_{\mathrm{head}} - m_1 \cdot i$
    id $\leftarrow$ GetPebbleId($p_i$)
    if $b = 1$ and ReadBit() $= 0$
        id $\leftarrow$ id $+ 2^j$
    else if $b = 0$ and ReadBit() $= 1$
        id $\leftarrow$ id $- 2^j$
    PutPebbleAtId($p_i$,id)
```

tape of $\mathcal{M}$ by changing the positions of the pebbles along ω. The transition function δ of $\mathcal{M}$ determines how $\mathcal{M}$ does computations on its tape and, in particular, how $\mathcal{M}$ changes its head position. We use a variable T_{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 T_{head} is sufficient to store the head position. In order to simulate one transition of $\mathcal{M}$ according to δ, 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 4). First, let us consider the two auxiliary functions GetPebbleId(p_i) and PutPebbleAtId(p_i, id) (cf. Algorithms 3). As the name suggests, the function GetPebbleId(p_i) returns the unique identifier associated to the vertex marked by p_i. Given an identifier id, we can use the function PutPebbleAtId(p_i, id) for placing pebble p_i at the unique vertex corresponding to id. By the choice of our encoding, if $T_{\mathrm{head}} = i \cdot m_1 + j$ with $j \in \{0, \ldots, m_1 - 1\}$, then bit j of the position of pebble p_i encodes the contents of the tape cell specified by T_{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. When $\mathcal{M}$ moves from a vertex v to another vertex v', the walk ω and the pebbles on it need to be relocated. Recall that $\mathcal{M}'$ marked the current vertex v with the pebble p_{start}. After having computed the label of the edge to v', $\mathcal{M}'$ drops the pebble p_{next} at v'. Then $\mathcal{M}'$ moves the pebbles placed along the walk ω to the corresponding positions along a new walk ω' starting at v' in the following way. We iterate over all $c - 3$ 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 p_{next} and place p_i on ω' using the function PutPebbleAtId(p_i, id). In this call of PutPebbleAtId(p_i, id) all occurrences of p_{start} are replaced by p_{next}. This way, we can carry the memory simulated by the pebbles along during the graph traversal.

Overall, we have shown that $\mathcal{M}'$ can simulate the traversal of $\mathcal{M}$ in G while using a tape with half the length, but c additional pebbles and a factor of c additional states. $\square$

Finally, we show that our recursive construction explores any graph on at most n vertices while using $\mathcal{O}(\log\log n)$ pebbles.

Theorem 3.4. *Any bounded-degree graph on at most n vertices can be explored using $\mathcal{O}(\log\log n)$ pebbles and memory.*

Proof. By Theorem 3.2, there are a constant $c \in \mathbb{N}$ and a $(c, 0, c\log n)$-pebble machine $\mathcal{M}$ exploring any bounded-degree graph on at most n vertices. Further, let c be larger than the constant from Theorem 3.3. We start with a $(c^{\log\log n}, c \cdot \log\log n, c)$ pebble machine $\mathcal{M}'$. Note that $\mathcal{M}'$ uses $\mathcal{O}(\log\log n)$ space and $\mathcal{O}(\log\log n)$ pebbles. Applying Theorem 3.3 exactly $\log\log n$ times yields that $\mathcal{M}'$ can simulate any $(c, 0, c \cdot 2^{\log\log n})$-pebble machine. As $c \cdot 2^{\log\log n} = c \cdot \log n$, we can in particular simulate $\mathcal{M}$. Therefore, $\mathcal{M}'$ can explore any bounded-degree graph on at most n vertices. $\square$

Corollary 3.1. *Our exploration algorithm can be adapted to eventually return to the starting vertex and terminate after $n^{\mathcal{O}(\log\log n)}$ steps, without additional memory or pebbles, even if the number of vertices of the graph is not known a priori.*

Proof (Sketch). The bound in Theorem 3.4 is attained by using $\log\log n$ levels of recursion which each require constant memory and a constant number of pebbles. Each level i of the recursion uses a closed walk with n_i distinct vertices to simulate additional memory. We know that we have visited all vertices once n_i does not increase any more, even if we do not know n a priori. We can thus safely terminate and return to the starting location by returning to the pebble p_{start} in every level of the recursion.

Let $\mathcal{M}$ be the adapted $(c^{\log\log n}, c\log\log n, c)$ pebble machine that explores any graph on at most n vertices, as described in Theorem 3.4, and additionally terminates and returns to the starting vertex. During the traversal, the position of the $\mathcal{O}(\log\log n)$ pebbles and the position and configuration of $\mathcal{M}$ cannot be twice the same, because $\mathcal{M}$ would be in a loop otherwise. There are $n^{\mathcal{O}(\log\log n)}$ possible positions of the pebbles, n for each pebble, and $\mathcal{O}(c^{\log\log n})$ configurations of the pebble machine $\mathcal{M}$. This means $\mathcal{M}$ must terminate after at most $n^{\mathcal{O}(\log\log n)}$ steps to not repeat its configuration and the position of the pebbles. □

4 Lower bounds for agents with pebbles

The goal in this section is to obtain a lower bound on the number of pebbles needed for exploring any graph on at most n vertices if the agent has less than $\mathcal{O}(\log n)$ bits of memory available. For our construction it will be convenient to think of an agent with p pebbles as a finite state automaton instead of a pebble machine. Given an (s', p, m)-pebble machine $\mathcal{M}$, we can construct a finite state automaton $\mathcal{A}$ with $s = s'm2^m$ states, one state for each of the $s'm2^m$ possible configurations of $\mathcal{M}$, with the same behavior. After observing the environment according to δ_{in}, the automaton directly transitions to the state corresponding to the configuration resulting from the computation of $\mathcal{M}$. We will call $\mathcal{A}$ an s-state agent with p pebbles.

In this section, we will build a *trap* for an arbitrary agent, i.e., a graph that the agent does not explore. The number of vertices of this trap yield a lower bound for the number of pebbles required for exploration, given that the agent has at most $\mathcal{O}((\log n)^{1-\epsilon})$ bits of memory for some constant $\epsilon > 0$. 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 $\{0, 1, 2\}$.

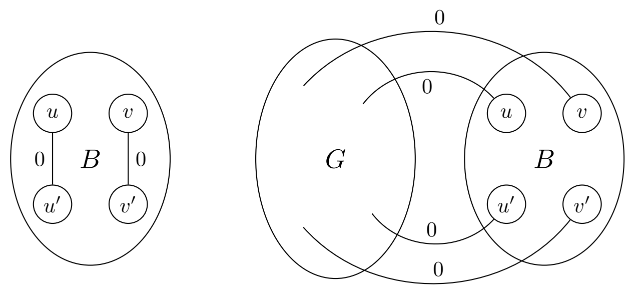

Figure 3: The r-barrier B on the left with two distinguished edges $\{u, u'\}$, $\{v, v'\}$ can be connected to an arbitrary graph G, as shown on the right.

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 an agent with too few pebbles. 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 $\{u, u'\}$ and $\{v, v'\}$ 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 $\{u, u'\}$ and $\{v, v'\}$ 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.

Definition 4.1. *For $r \leq p$, the graph B is an* r-barrier *for an agent $\mathcal{A}$ with p pebbles if, in the above setting, the following holds for all graphs G and every pair (a, b) in $\{v, v'\} \times \{u, u'\}$: If $\mathcal{A}$ starts in an arbitrary vertex in G, then it never traverses B from a to b or vice versa with a set of at most r pebbles carried by the agent, observed or placed on vertices of B during the traversal. We equivalently say that $\mathcal{A}$ cannot traverse B from a to b or vice versa while* using at most r pebbles.

A p-barrier immediately yields a trap for the agent.

Lemma 4.1. *Given a p-barrier with m vertices, we can construct a trap with $2m + 4$ vertices.*

Proof. Let H_1 and H_2 be two copies of a p-barrier for agent $\mathcal{A}$. We connect the two graphs and four additional

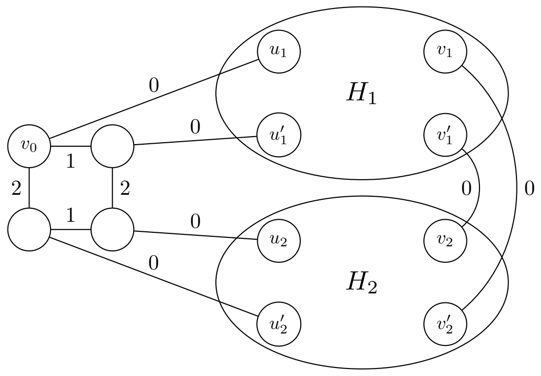

Figure 4: Constructing a trap given p-barrier B.

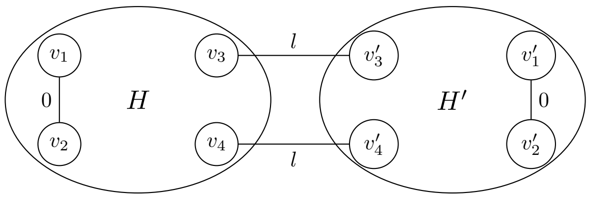

Figure 5: Graph B with the property that $\mathcal{A}$ never visits v'_1 or v'_2 when starting in any state in v_1 or v_2 and vice versa.

vertices, as shown in Figure 4. If the agents start in vertex v_0, then it cannot reach v_1 or v'_1 via the p-barrier H_1 or via the p-barrier H_2. Thus $\mathcal{A}$ does not explore the graph. The constructed trap for $\mathcal{A}$ contains $2m + 4$ vertices. □

Our goal for the remaining section is to construct a p-barrier for a given agent $\mathcal{A}$ and give a good upper bound on the number of vertices it contains, because this will give an upper bound for the number of vertices of a trap for $\mathcal{A}$ by Lemma 4.1. The construction of the p-barrier is recursive. We start with a 0-barrier which builds on the following useful result stating that, for any set of q agents without pebbles, there is a graph containing an edge which is not traversed by any of them.

Theorem 4.1. ([16, Theorem 4]) *For any q non-cooperative s-state agents without pebbles, there exists a 3-regular graph G on $\mathcal{O}(qs)$ vertices with the following property: There are two edges $\{v_1, v_2\}$ and $\{v_3, v_4\}$ in G, the first labeled 0, such that any of the q agents starting in v_1 or v_2 does not traverse the edge $\{v_3, v_4\}$.*

Lemma 4.2. *For every s-state agent with p pebbles $\mathcal{A}$ there exists a 0-barrier with $\mathcal{O}(s^2)$ vertices, which is independent of the starting state of $\mathcal{A}$.*

Proof. Let Q be the set of states of the agent $\mathcal{A}$ and q_0 its starting state. For all $q \in Q$, we define the agent without pebbles $\mathcal{A}_q$ to be the agent with the same behavior as $\mathcal{A}$, but without pebbles and starting in state q instead of q_0. That is, $\mathcal{A}_q$ has the same set of states as $\mathcal{A}$ and it transitions according to the functions δ_{in} and δ_{out} of $\mathcal{A}$, while the set of pebbles carried and observed is always empty. Moreover, let $S := \{\mathcal{A}_q \mid q \in Q\}$.

Theorem 4.1 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 $\mathcal{A}_q$ that starts in v_1 or v_2 does not traverse the edge $\{v_3, v_4\}$. Let the graph B consist of two connected copies of H, as illustrated in Figure 5. The edges $\{v_3, v_4\}$ and $\{v'_3, v'_4\}$ are replaced by $\{v_3, v'_3\}$ and $\{v_4, v'_4\}$, which are also labeled l. We claim that B is a 0-barrier for $\mathcal{A}$.

For the sake of contradiction, assume that there is a graph G that can be connected to B so that starting in G in any state, the agent $\mathcal{A}$ would without loss of generality walk from v_1 to v'_1 in B without using any pebbles. Then $\mathcal{A}$ starts this walk in a state q in v_1 and the traversal sequence of $\mathcal{A}$ is the same as that of $\mathcal{A}_q$ starting in v_1. This would mean that $\mathcal{A}_q$ traverses the edge $\{v_3, v_3\}$, which contradicts Theorem 4.1. □

The proof of Theorem 4.1 uses the fact that the next state of an agent without pebbles only depends on the previous state when traversing a regular graph. Thus, after at most s steps, the state of the agent and therefore also the next label chosen need to repeat with a period of length at most s. For an agent with pebbles, however, the next state and label that are chosen may also depend on the sets of pebbles the agent observes and carries. Hence, different relative positions of the pebbles around it can eventually lead to different states when they are encountered. We therefore need to account for the positions of all pebbles when forcing the agent into a periodic behavior. This motivates the following definition.

Definition 4.2. *The* configuration *of an s-state agent $\mathcal{A}$ with p pebbles in a vertex v is a $(p+1)$-tuple $(q, P_1, P_2, \ldots, P_p)$, where q is the current state of $\mathcal{A}$, and P_i is the position of the i-th pebble relative to the agent. Formally, we can define P_i to be the shortest traversal sequence leading from the location of the agent to the i-th pebble. If there is more than one, we choose the lexicographically smallest traversal sequence. We set $P_i = \emptyset$ if the pebble is at v and $P_i = (-1)$, if the pebble is carried by $\mathcal{A}$.*

In order to limit the number of possible configurations, we force the agent to keep all p pebbles close by.

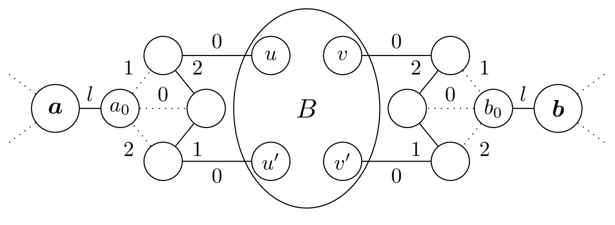

Figure 6: An edge $\{a, b\}$ labeled l is replaced with the gadget $B(l)$ containing an r-barrier B. Only the dotted edges incident to a_0 and b_0 that are not labeled l are part of the gadget. Consequently, the gadget contains two vertices of degree 2. The vertices a and b are macro vertices of a graph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

Intuitively, we can achieve this for any graph G by replacing all edges with $(p-1)$-barriers. This way, the agent cannot move between neighboring vertices of the original graph G without taking all p pebbles along. 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 4.3. *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 and an r-barrier B for an agent $\mathcal{A}$ with $p \geq r$ pebbles, we replace edges of G using the following construction. First, we replace every edge $\{a, b\}$ labeled l with the gadget $B(l)$ shown in Figure 6, and we call the resulting graph $G_1(B)$. By construction, the labels of the edges incident to the same vertex in $G_1(B)$ are distinct. However, certain vertices only have degree 2. We take the 3-regular extension of $G_1(B)$ and define the resulting graph as $G(B) := \overline{G_1(B)}$.

The graph $G(B)$ contains two copies of $G_1(B)$. 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 now establish that the agent has to keep all pebbles close by.

Lemma 4.3. *Let B be a $(p-1)$-barrier for an agent $\mathcal{A}$ with p pebbles. Then, the following hold for any graph of the form $G(B)$.*

1. *$\mathcal{A}$ cannot get from a macro vertex v to a distinct macro vertex v' while using less than p pebbles.*
2. *At any time, there is some macro vertex v such that $\mathcal{A}$ and each pebble are at v or in one of the surrounding gadgets $B(0)$, $B(1)$ and $B(2)$.*

Proof. For the sake of contradiction, assume that $\mathcal{A}$ walks from a macro vertex v to a distinct macro vertex v' with less than p pebbles. The graph $G(B)$ contains two copies of $G_1(B)$, but all vertices in the $(p-1)$-barriers within $G_1(B)$ have degree 3. Thus, $\mathcal{A}$ must have walked through some $(p-1)$-barrier B with less than p pebbles. This contradicts that B is a $(p-1)$-barrier. Therefore, $\mathcal{A}$ needs to use all pebbles to get from a macro vertex v to a distinct macro vertex v'.

For the second part of the claim, consider the positions of the pebbles and $\mathcal{A}$ after an arbitrary number of steps and let v be the macro vertex last visited by $\mathcal{A}$. By the first part of the claim, all pebbles are either carried by $\mathcal{A}$, located at v, or at any of the three surrounding gadgets. ☐

Let B be a $(p-1)$-barrier for an agent $\mathcal{A}$ with p pebbles and $\mathcal{A}$ start in some macro vertex v_0 of $G(B)$. Iteratively, define $t_0 = 0$ and t_i to be the first point in time after t_{i-1}, when $\mathcal{A}$ visits a macro vertex v_i distinct from v_{i-1} (and its copy v'_{i-1}). Then, the sequence $v_0, v_1, \dots$ corresponds to a path in G. This sequence of neighboring vertices in G yields a unique sequence of labels $l_0, l_1, \dots$ of the edges between the neighboring vertices in G, which we call the *macro traversal sequence* of $\mathcal{A}$ starting in vertex v_0 in $G(B)$. Note that the macro traversal sequence may be finite.

Consider the traversal sequence $l_0, l_1, \dots$ of an agent without pebbles in a 3-regular graph G and the traversal sequence $l'_0, l'_1, \dots$ in another 3-regular graph G'. If the state of the agent in G after i steps is the same as the state in G' after j steps, then the traversal sequences coincide from that point on, i.e., $l_{i+k} = l'_{j+k}$ holds for all $k \in \mathbb{N}$. The reason is that every vertex looks exactly the same and thus the next state of the agent only depends on the previous one. We want to obtain a similar result for agents with pebbles. However, in general it is not true that if the configuration of an agent in a graph G after i steps is the same as after j steps in G', then the next configurations and chosen labels coincide. This is because the pebbles give the agent the ability to detect differences in the graphs G and G'. The agent could, for example, drop a pebble and walk in a loop that is only part of one of the graphs and this may lead to different

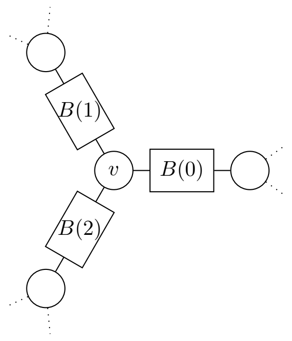

Figure 7: A macro vertex v in a graph $G(B)$ surrounded by the three gadgets $B(1), B(2)$ and $B(3)$.

configurations. That is why we consider graphs of the form $G(B)$. In these graphs, all macro vertices look the same, as they are surrounded by the same gadgets, and the pebbles are always close to the agent, making it impossible for the agent to detect a loop that is part of one of the graphs, but not the other. This intuition is formally expressed in the next lemma.

Lemma 4.4. *Let B be a $(p-1)$-barrier for an agent $\mathcal{A}$, G and G' be two 3-regular graphs, $v_0, v_1, \ldots$ the sequence of macro vertices visited by $\mathcal{A}$ in $G(B)$ and $l_0, l_1, \ldots$ be the corresponding macro traversal sequence. Similarly, let $v'_0, v'_1, \ldots$ be the sequence of macro vertices visited by $\mathcal{A}$ in $G'(B)$ and $l'_0, l'_1, \ldots$ be the corresponding macro traversal sequence. If at some point the configuration of $\mathcal{A}$ in v_i is the same as the configuration of $\mathcal{A}$ in v'_j, then $l_{i+k} = l'_{j+k}$ holds for all $k \in \mathbb{N}$.*

Proof. Let t_i and t'_j be such that the configuration of $\mathcal{A}$ after t_i steps in v_i is the same as after t'_j steps in v'_j. Iteratively, define t_a for $a > i$ to be the first time after t_{a-1} that the agent visits v_a and analogously t'_b for $b > j$ the first time after t'_{b-1} that the agent visits v'_b.

By induction, we show that for $k \in \mathbb{N}$ we have $l_{i+k} = l'_{j+k}$ and the configuration of $\mathcal{A}$ after t_{i+k} steps in v_{i+k} is the same as the configuration of $\mathcal{A}$ after t'_{j+k} steps in v'_{j+k}. Intuitively, this means that the agent will first reach the macro vertex v_{i+k} in the same configuration as it reaches v'_{j+k}. The idea of the proof is that in between visits to macro vertices, the agent behaves the same in the two graphs and, in particular, it traverses the same gadget $B(l)$ in both cases so that $l_{i+k} = l'_{j+k}$.

For $k = 0$ the claim holds by assumption. Now, assume that it holds for some $k \in \mathbb{N}$. The graphs $G(B)$ and $G'(B)$ locally look the same to the agent in v_{i+k} and v'_{j+k}, as both macro vertices 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 $G(B)$ containing v_{i+k} and all surrounding gadgets to the induced subgraph of $G'(B)$ containing v'_{j+k} and all surrounding gadgets. Moreover, γ respects the labeling and maps v_{i+k} to v'_{j+k}. As the configuration of $\mathcal{A}$ after t_{i+k} steps in v_{i+k} is the same as the configuration of $\mathcal{A}$ after t'_{j+k} steps in v'_{j+k}, the isomorphism also respects the position of the pebbles. As v_{i+k+1} is the first macro vertex visited after v_{i+k}, all pebbles are at v_{i+k} or any of the surrounding gadgets until $\mathcal{A}$ reaches v_{i+k+1} by Lemma 4.3. The same holds for v'_{j+k} and v'_{j+k+1}. Iteratively, for $t = 0, 1, \ldots$ the following holds until the agent reaches the next macro vertex v_{i+k+1} or v'_{j+k+1}.

1. The state of $\mathcal{A}$ after $t_{i+k} + t$ steps in $G(B)$ is the same as the state of $\mathcal{A}$ after $t'_{j+k} + t$ steps in $G'(B)$.

2. The isomorphism γ maps the position of each pebble i after $t_{i+k} + t$ steps in $G(B)$ to the position of the pebble i after $t'_{j+k} + t$ in $G'(B)$.

3. The isomorphism γ maps the position of $\mathcal{A}$ after $t_{i+k} + t$ steps in $G(B)$ to the position of $\mathcal{A}$ after $t'_{j+k} + t$ in $G'(B)$.

This implies that v_{i+k} and v_{i+k+1} are connected with the same gadget as v'_{j+k} and v'_{j+k+1}, i.e., $l_{i+k+1} = l'_{j+k+1}$. Furthermore, there is $\bar{t}$ such that $t_{i+k+1} = t_{i+k} + \bar{t}$ and $t'_{j+k+1} = t'_{j+k} + \bar{t}$. Moreover, the configuration of $\mathcal{A}$ in v_{i+k+1} after t_{i+k+1} steps is the same as in v'_{j+k+1} after t'_{j+k+1} steps. $\square$

For constructing an r-barrier B' for an agent $\mathcal{A}$ with p pebbles given an $(r-1)$-barrier B, we need to examine the behavior of $\mathcal{A}$, when only using a subset of r pebbles. There are $\binom{p}{r}$ subsets of size r of the pebbles and $\mathcal{A}$ may behave differently depending on which subset it is using. We denote these $\binom{p}{r}$ subsets of r pebbles with $M_r^{(1)}, \ldots, M_r^{\binom{p}{r}}$ and further let $\mathcal{A}_r^{(k)}$ denote the s-state automaton with the same behaviour as $\mathcal{A}$ when only equipped with the set of r pebbles $M_r^{(k)}$. That is, $\mathcal{A}_r^{(k)}$ has the same set of states as $\mathcal{A}$ and it transitions according to the functions δ_{in} and δ_{out} of $\mathcal{A}$, while the set of pebbles carried and observed is always a subset of $M_r^{(k)}$.

Furthermore, let α_B be the maximum number of possible configurations of $\mathcal{A}_r^{(k)}$ in a macro vertex in a graph of the form $G(B)$. Note that this definition is independent of k and G, because B is an $(r-1)$-barrier and all pebbles are either carried by the agent, placed on the current macro vertex or in any of the surrounding gadgets $B(0), B(1), B(2)$ in any graph $G(B)$ by Lemma 4.3.

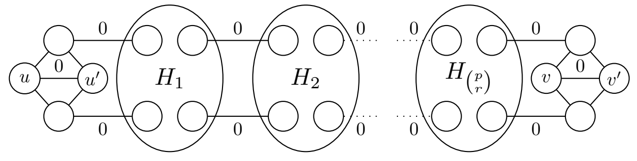

Figure 8: Connecting the graphs $H_1, H_2, \ldots, H_{\binom{p}{r+1}}$ to a graph H, yields the $(r+1)$-barrier $H(B)$.

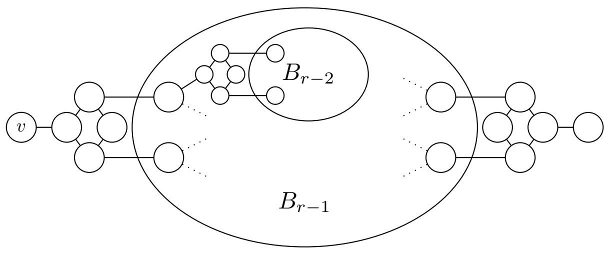

Figure 9: Recursive structure of an r-barrier.

We can now present the construction of an r-barrier given an $(r-1)$-barrier.

THEOREM 4.2. *Given an $(r-1)$-barrier B with m vertices for an agent $\mathcal{A}$ with $p \geq r$ pebbles, we can construct an r-barrier B' with $\mathcal{O}(\binom{p}{r} \cdot m \cdot \alpha_B^2)$ vertices for $\mathcal{A}$.*

Proof. For $k \in \{1, 2, \ldots, \binom{p}{r}\}$, the possible configurations of $\mathcal{A}_r^{(k)}$ in macro vertex of a graph of the form $G(B)$ can be enumerated $x_1, \ldots, x_{\alpha_B}$. By Lemma 4.4, the configuration of $\mathcal{A}_r^{(k)}$ uniquely determines the next label in the macro label sequence of $\mathcal{A}_r^{(k)}$ independently of the underlying graph G. We can therefore define the following agent $\mathcal{B}^{(k)}$ without pebbles: The set of states of $\mathcal{B}^{(k)}$ is $\{q_1, \ldots, q_{\alpha_B}\}$. Moreover, in state q_i the agent $\mathcal{B}^{(k)}$ traverses the edge labeled l and transitions to q_j, if $\mathcal{A}_r^{(k)}$ in configuration x_i will traverse the gadget $B_r(l)$ (i.e. l is the next label in the macro label sequence of $\mathcal{A}_r^{(k)}$ in configuration x_i). The starting state of $\mathcal{B}^{(k)}$ is the configuration x_i, where $\mathcal{A}_r^{(k)}$ carries the whole set of pebbles $M_r^{(k)}$. Note that this is well-defined because of Lemma 4.4 and, in particular, the macro traversal sequence of $\mathcal{A}_r^{(k)}$ in $G(B)$ is exactly the same as the traversal sequence of $\mathcal{B}^{(k)}$ in G independently of the graph G.

By Lemma 4.2, there is a 0-barrier H_k with $\mathcal{O}(\alpha_B^2)$ vertices that cannot be traversed by $\mathcal{B}^{(k)}$ for any starting state. We connect the graphs $H_1, H_2, \ldots, H_{\binom{p}{r+1}}$, as shown in Figure 8, and let H be the resulting graph. The claim is that $B' := H(B)$ is an r-barrier for $\mathcal{A}$.

For the sake of contradiction, assume that there is a subset of r pebbles M_k and some graph G connected to $H(B)$ such that $\mathcal{A}$ can traverse $H(B)$ from u to v using the set of pebbles M_k. This means that $\mathcal{A}$ traversed the induced subgraph $H_k(B)$ using the set of pebbles M_k. But then $\mathcal{A}_r^{(k)}$ can traverse $H_k(B)$ for a suitable starting configuration implying that $\mathcal{B}^{(k)}$ can traverse H_k for a suitable starting state. This is a contradiction.

Finally, each H_k contains $\mathcal{O}(\alpha_B^2)$ vertices and therefore H has at most $\mathcal{O}(\binom{p}{r}\alpha_B^2)$ vertices. As B has m vertices, the number of vertices of $B' = H(B)$ is at most $\mathcal{O}(\binom{p}{r} \cdot m \cdot \alpha_B^2)$, where we use that H is 3-regular and therefore the number of edges is 3/2 times the number of vertices. □

We now fix an agent $\mathcal{A}$ with p pebbles and let B_0 be the 0-barrier given by Lemma 4.2 and B_r for $0 < r \leq p$ be the r-barrier constructed recursively using Theorem 4.2. Moreover, we let m_r be the number of vertices of B_r and $\alpha_r := \alpha_{B_{r-1}}$ be the maximum number of possible configurations of $\mathcal{A}_r^{(k)}$ in a macro vertex in a graph of the form $G(B_{r-1})$.

We want to bound the number of vertices m_p of B_p and thus, according to Lemma 4.1, also the number of vertices of the trap for $\mathcal{A}$. By Theorem 4.2, there is a constant $c \in \mathbb{N}$ such that $m_r \leq c\binom{p}{r} \cdot m_{r-1} \cdot \alpha_r^2$. In order to bound m_r, we therefore need to bound the number of configurations α_r. We can obtain a bound from Lemma 4.3, since the pebbles are never far from the agent's location. For a tight bound in our main result, however, we need a careful analysis of the recursive structure of our construction. The idea is that there cannot be a set of i pebbles which is separated by an $(r-i)$-barrier from the agent's location. This yields the following bound on α_r.

LEMMA 4.5. *The maximum number of possible configurations α_r of $\mathcal{A}_r^{(k)}$ in a macro vertex in a graph of the form $G(B_{r-1})$ can be bounded by $s \cdot r! \prod_{i=1}^{r}(2^{3i} \cdot m_{r-i} + 2^{3i+1})$.*

Proof. If we consider a macro vertex v in a graph $G(B_{r-1})$ and look into the recursive structure of the barrier B_{r-1}, as shown in Figure 9, we note that it contains barriers B_{r-2}, which, in turn, contain barriers B_{r-3} and so on.

For $i < r$, we call a vertex w i-adjacent to v if there is a path from v to w that does not traverse an i-barrier B_i. As a convention, v itself is i-adjacent to v. Note that a vertex w contained in an i-barrier may be i-adjacent if there is a path from v to w that does not traverse a distinct i-barrier.

First, we bound the number of vertices that are i-adjacent to v for all $i \in \{0, 1, \ldots, r-1\}$. Observe that the distance from v to any i-adjacent vertex, which is not contained in a barrier B_i, is at most $3(r-i)$.

This observation is clear for $i = r-1$ and follows for $r-2, r-3, \dots$ by examining the recursive structure given in Figure 9. As $G(B_{r-1})$ is 3-regular, there are at most $2^{3(r-i)+1} - 1$ such vertices. Moreover, any i-barrier B_i containing vertices that are i-adjacent to v, in particular contains a vertex with a distance of exactly $3(r-i)$ to v. As $G(B_{r-1})$ is 3-regular, there are at most $2^{3(r-i)}$ vertices of distance exactly $3(r-i)$ from v and therefore at most $2^{3(r-i)}$ different i-barriers with m_i vertices containing i-adjacent vertices. Thus, there are at most $2^{3(r-i)} \cdot m_i$ vertices that are i-adjacent to v and contained in a barrier B_i. Overall, there are at most $2^{3(r-i)} \cdot m_i + 2^{3(r-i)+1} - 1$ vertices i-adjacent to v.

When the agent is currently at the macro vertex v, we know that all pebbles are carried by the agent or placed on an $(r-1)$-adjacent vertex by Lemma 4.3. But we can show the following stronger statement: For all $i \in \{0, \dots, r-1\}$, at least $i+1$ pebbles are carried or placed on a vertex that is i-adjacent to v.

The claim for $i = r-1$ follows from Lemma 4.3. We show the claim for $i = r-2$. The proof is analogous for any other value of i. For the sake of contradiction, assume that there are two pebbles which are placed on vertices which are not $(r-2)$-adjacent. Then the agent at v and the remaining set of at most $r-2$ pebbles are separated by barriers B_{r-2} from the two pebbles. But this is a contradiction, as $\mathcal{A}$ could not have crossed B_{r-2} with at most $r-2$ pebbles.

As a vertex that is i-adjacent is also $(i+1)$-adjacent, we have shown that there exists an enumeration of the pebbles $a_0, \dots, a_{r-1}$ such that pebble a_i is placed on a vertex that is i-adjacent to v for $i \in \{0, \dots, r-1\}$. We already showed that the number of vertices that are i-adjacent to v is at most $2^{3(r-i)} \cdot m_i + 2^{3(r-i)+1} - 1$. Pebble a_i is either carried by the agent or placed on an i-adjacent vertex and thus there are at most $2^{3(r-i)} \cdot m_i + 2^{3(r-i)+1}$ possibilities for a_i. Overall, there are $r!$ possible enumerations of pebbles, s possible states of the agent and therefore the number of configurations of $\mathcal{A}$ in a macro vertex v can be bounded by $s \cdot r! \prod_{i=0}^{r-1} (2^{3(r-i)} \cdot m_i + 2^{3(r-i)+1})$. A change of index yields the claim. □

Using the bound on α_r from Lemma 4.5, we can bound the number of vertices of the barriers.

THEOREM 4.3. *For $r \leq p$ and $s \geq 2^p$, the number of vertices of the r-barrier B_r for the s-state agent $\mathcal{A}$ with p pebbles is bounded by $\mathcal{O}(s^{8^{r+1}})$.*

Proof. The existence of an r-barrier follows from Lemma 4.2 and Theorem 4.2. Therefore, we only need to bound the number of vertices of the barriers. Using $r! \leq r^r \leq s^r$ for $r > 0$ and $(2^{3i} \cdot m_{r-i} + 2^{3i+1}) \leq 2^{3i+1} m_{r-i}$ as $m_{r-i} \geq 2$, we can simplify the bound from Lemma 4.5 to $\alpha_r \leq s^{r+1} 2^{\sum_{i=1}^{r}(3i+1)} \prod_{i=0}^{r-1} m_i$. We have $\sum_{i=1}^{r}(3i+1) = 3/2(r^2+r)+r \leq 2r^2+3r$ and $2^r \leq 2^p \leq s$ and obtain

$$\alpha_r \leq s^{r+1} 2^{2r^2+3r} \prod_{i=0}^{r-1} m_i \leq s^{3r+4} \prod_{i=0}^{r-1} m_i.$$

Moreover, by Lemma 4.2 and Theorem 4.2, there is a constant $c \in \mathbb{N}$ such that

$$m_0 \leq cs^2 \quad \text{and} \quad m_r \leq c\binom{p}{r} \cdot m_{r-1} \cdot \alpha_r^2.$$

For the asymptotic bound, we may assume $s \geq c$. We have $\binom{p}{r} \leq 2^p \leq s$ and plugging in the bound for α_r, the above inequalities yield $m_0 \leq s^3$ and

$$m_r \leq s^2 m_{r-1} \left(s^{3r+4} \prod_{i=0}^{r-1} m_i \right)^2 = s^{6r+10} m_{r-1} \prod_{i=0}^{r-1} m_i^2.$$

It remains to show that $m_r \leq s^{8^{r+1}}$ for all $r \leq p$ by induction on r. We have $m_0 \leq s^3 \leq s^8$. Now, assume $m_{r-1} \leq s^{8^r}$ holds. From the above inequality we obtain

$$m_r \leq s^{6r+10} \cdot s^{8^r} \prod_{i=0}^{r-1} s^{2 \cdot 8^{i+1}} = s^{6r+10+8^r+2\cdot\sum_{i=1}^{r} 8^i}.$$

Thus, we only need to bound the exponent. We have $6r+10 \leq 2 \cdot 8^r$ for $r \geq 1$ and further $2 \cdot \sum_{i=1}^{r} 8^i \leq 2/7 \cdot 8^{r+1} \leq 3 \cdot 8^r$. Overall, the exponent can therefore be bounded by 8^{r+1}, as desired. □

The bound on the number of vertices of a trap for an agent with pebbles follows directly from Theorem 4.3 and Lemma 4.1.

THEOREM 4.4. *For any s-state agent with p pebbles, with $s \geq 2^p$, there exists a trap with at most $\mathcal{O}(s^{8^{p+1}})$ vertices.*

Finally, we derive a bound on the number of pebbles needed for exploration if the agent has less than $\mathcal{O}(\log n)$ bits of memory available. Note that this bound also holds for indistinguishable pebbles as these are less powerful.

THEOREM 4.5. *For any constant $\varepsilon > 0$, an agent with at most $\mathcal{O}((\log n)^{1-\varepsilon})$ bits of memory needs at least $\Omega(\log\log n)$ distinguishable pebbles for exploring all graphs on at most n vertices.*

Proof. Let $\varepsilon > 0$ be constant and $\mathcal{A}$ an agent with p pebbles and $\mathcal{O}((\log n)^{1-\varepsilon})$ 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 $\mathcal{A}$ has $s := 2^{c\cdot(\log n)^{1-\varepsilon}}$ states. If $s < 2^p$, then the agent already asymptotically uses at least $\Omega(\log\log n)$ distinguishable pebbles and we are done. Otherwise, we can apply Theorem 4.4 and obtain a trap for $\mathcal{A}$ containing $\mathcal{O}(s^{8^{p+1}})$ vertices. As $\mathcal{A}$ explores any graph on at most n vertices, we have $n \leq \mathcal{O}(1)s^{8^{p+1}}$. By taking logarithms on both sides of this inequality, we obtain

$$\log n \leq \mathcal{O}(1) + 8^{p+1}\log s = \mathcal{O}(1) + 8^{p+1}c(\log n)^{1-\varepsilon}.$$

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

Acknowledgement

We would like to thank the anonymous reviewers for their careful reading and many useful comments, in particular for pointing out how to strengthen Theorem 4.5.

References

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

[2] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovasz, and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *Proc. 20th Annu. IEEE Symp. Found. Comput. Sci. (FOCS)*, pages 218–223, 1979.

[3] C. Ambühl, L. Gąsieniec, A. Pelc, T. Radzik, and X. Zhang. Tree exploration with logarithmic memory. *ACM Trans. Algorithms*, 7(2):1–21, 2011.

[4] B. Awerbuch, M. Betke, R. L. Rivest, and M. Singh. Piecemeal graph exploration by a mobile robot. *Inform. and Comput.*, 152(2):155–172, 1999.

[5] M. A. Bender, A. Fernández, D. Ron, A. Sahai, and S. Vadhan. The Power of a Pebble: Exploring and Mapping Directed Graphs. *Information and Computation*, 176(1):1 – 21, 2002.

[6] M. Blum and C. Hewitt. Automata on a 2-dimensional tape. In *Proc. 8th Annu. Symp. Switching and Automata Theory (FOCS)*, pages 155–160, 1967.

[7] M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *Proc. 19th Annu. IEEE Symp. Found. Comput. Sci. (FOCS)*, pages 132–142, 1978.

[8] M. Blum and W. J. Sakoda. On the capability of finite automata in 2 and 3 dimensional space. In *Proc. 18th Annu. IEEE Symp. Found. Comput. Sci. (FOCS)*, pages 147–161, 1977.

[9] J. Chalopin, S. Das, Y. Disser, M. Mihalák, and P. Widmayer. Mapping simple polygons: How robots benefit from looking back. *Algorithmica*, 65(1):43–59, 2013.

[10] J. Chalopin, S. Das, Y. Disser, M. Mihalák, and P. Widmayer. Mapping simple polygons: The power of telling convex from reflex. *ACM Trans. Algorithms*, 11(4):1–16, 2015.

[11] J. Chalopin, S. Das, and A. Kosowski. Constructing a map of an anonymous graph: Applications of universal sequences. In *Proc. 14th Int. Conf. Principles Distributed Systems (OPODIS)*, pages 119–134, 2010.

[12] K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree exploration with little memory. *J. Algorithms*, 51(1):38 – 63, 2004.

[13] G. Dudek, M. Jenkin, E. E. Milios, and D. Wilkes. Robotic exploration as graph construction. *IEEE Trans. Robotics Automation*, 7(6):859–865, 1991.

[14] P. Fraigniaud and D. Ilcinkas. Digraphs exploration with little memory. In *Proc. 21st Annu. Sympos. Theoretical Aspects Comput. Sci. (STACS)*, pages 246–257, 2004.

[15] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoret. Comput. Sci.*, 345(2–3):331–344, 2005.

[16] P. Fraigniaud, D. Ilcinkas, S. Rajsbaum, and S. Tixeuil. The reduced automata technique for graph exploration space lower bounds. *Theoretical Computer Science. Essays in Memory of Shimon Even*, pages 1–26, 2006.

[17] F. Hoffmann. One pebble does not suffice to search plane labyrinths. In *Proc. 3rd Int. Symp. Fundamentals of Computation Theory (FCT)*, pages 433–444, 1981.

[18] S. Hoory and A. Wigderson. Universal traversal sequences for expander graphs. *Inform. Process. Lett.*, 46(2):67–69, 1993.

[19] S. Istrail. Polynomial universal traversing sequences for cycles are constructible. In *Proc. 20th Annu. ACM Symp. Theory Computing (STOC)*, pages 491–503, 1988.

[20] M. Koucký. Universal traversal sequences with backtracking. *J. Comput. System Sci.*, 65(4):717–726, 2002.

[21] M. Koucký. Log-space constructible universal traversal sequences for cycles of length $\mathcal{O}(n^{4.03})$. *Theoret. Comput. Sci.*, 296(1):117–144, 2003.

[22] M. Koucký. *On Traversal Sequences, Exploration Sequences and Completeness of Kolmogorov Random Strings*. PhD thesis, Rutgers, The State University of New Jersey, 2003.

[23] S. M. Mirtaheri, M. E. Dinçktürk, S. Hooshmand, G. V. Bochmann, G.-V. Jourdan, and I. V. Onut. A brief history of web crawlers. *arXiv preprint arXiv:1405.0749*, 2014.

[24] O. Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4):17, 2008.

[25] O. Reingold, L. Trevisan, and S. P. Vadhan. Pseudorandom walks on regular digraphs and the RL vs. L problem. In *Proc. 38th Annu. ACM Symp. Theory Computing (STOC)*, pages 457–466, 2006.

[26] H. Rollik. Automaten in planaren Graphen. *Acta Inform.*, 13:287–298, 1980.

[27] A. N. Shah. Pebble automata on arrays. *Comput. Vision Graph.*, 3(3):236–246, 1974.