The Space Complexity of Undirected Graph Exploration

Yann Disser[1(✉)] and Max Klimm[2]

[1] TU Darmstadt, Darmstadt, Germany
disser@mathematik.tu-darmstadt.de
[2] TU Berlin, Berlin, Germany
klimm@tu-berlin.de

Abstract. We review the space complexity of deterministically exploring undirected graphs. We assume that vertices are indistinguishable and that edges have a locally unique color that guides the traversal of a space-constrained agent. The graph is considered to be explored once the agent has visited all vertices. We visit results for this setting showing that $\Theta(\log n)$ bits of memory are necessary and sufficient for an agent to explore all n-vertex graphs. We then illustrate that, if agents only have sublogarithmic memory, the number of (distinguishable) agents needed for collaborative exploration is $\Theta(\log\log n)$.

Keywords: Graph exploration · Multi-agent exploration · Space complexity · Connectivity · Log-space

1 Introduction

When working with large data sets it is no longer justified to assume the entire input, or even a significant fraction of it, to be accessible at once. In particular, data may be spatially distributed along a dynamic network structure, such as the Internet or social networks. In this setting, the systematic navigation or crawling of the network becomes an integral component of any algorithmic processing of the data it holds. The theoretical framework of graph exploration is concerned precisely with the algorithmic problem of systematically traversing an initially unknown graph.

Generally, the main questions in graph exploration are regarding *feasibility*, i.e.,how much computational power is necessary for systematic exploration, and regarding *efficiency*, i.e.,how quickly a graph can be explored algorithmically. In the context of dealing with large data sets, the feasibility question is of particular importance. The necessary computational power can be captured theoretically by the space complexity of the exploration problem. Intuitively, the question is what portion of a graph we need to be able to memorize in order to avoid running in circles.

In this chapter, we review the most important results regarding the space complexity of undirected graph exploration. In Sect. 2, we introduce the graph exploration framework in more detail. In Sect. 3, we outline a general lower bound on the space complexity of graph exploration of $\Omega(\log n)$. Reingold's algorithm for undirected graph

H. Bast et al. (Eds.): Algorithms for Big Data, LNCS 13201, pp. 152–166, 2022.
https://doi.org/10.1007/978-3-031-21534-6_8

exploration is presented in Sect. 4. We then turn to collaborative graph exploration by a set of agents. In Sect. 5, we show that when all agents have sub-logarithmic memory $\mathcal{O}(\log^{1-\varepsilon} n)$ for some $\varepsilon > 0$, then $\Omega(\log\log n)$ agents are needed to explore any undirected graph with n vertices. Finally, in Sect. 6, we provide a matching upper bound showing that a team of $\mathcal{O}(\log\log n)$ agents can explore deterministically any undirected n-vertex graph, even if each agent has only constant memory.

The aim of this chapter is to survey the key ideas of these results, and we only sketch proofs on a high level. Whenever possible, intuition is prefered over formal statements, and many details are omitted to increase accessibility. For a more formal treatment, we refer to the original papers. Pointers to the relevant literature are given in Sect. 7.

2 Exploration and Feasibility

In the following, we consider an agent initially located at a vertex v_0 of an unknown, edge-colored, undirected graph $G = (V, E)$. We assume the edge-coloring to be locally unique in the sense that no two edges incident to a common vertex may share a color. The agent's perception of G is limited to observing the set of colors of the edges incident to its current location. In every step, the agent may choose one of these colors and move to the other endpoint of the corresponding edge. Importantly, vertices with the same set of colors adjacent to them are indistinguishable to the agent. The objective of the agent is to explore G, i.e.,to systematically visit all vertices of G in a finite number of steps. We are looking for a *deterministic* traversal algorithm that guarantees to explore every undirected graph. Regarding *randomized* traversal algorithms, it is known that a random walk of length $n^5 \log n$ visits all vertices of any graph with n vertices with high probability (Aleliunas et al. [1]). This yields a constant-space *perpetual* randomized graph exploration algorithm, i.e., an algorithm that runs forever and eventually visits all vertices. If n is known, combining this algorithm with a counter counting up to $n^5 \log n$ yields a log-space randomized graph exploration algorithm.

To illustrate the difficulty of deterministic exploration in this weak agent model, consider the exploration of a *fully regular* graph G, i.e.,a graph where all vertices are incident to edges of the exact same set of colors (cf. Fig. 1). Even if the agent knows that G is fully regular, after the first step where it learns the degree of the graph, its observations contain no information at all. In particular, every deterministic exploration algorithm must produce the same sequence of colors for any two fully regular graphs using the same colors. Intuitively, this is the most challenging setting for exploration. Then, the algorithmic problem reduces to asking for a *universal traversal sequence*, i.e.,a sequence of colors that we can follow to eventually visit all vertices, irrespective of G and v_0. Here and throughout, following a color sequence means to perform a sequence of movement decisions according to it, and we say that a color sequence explores G if the agent visits all vertices when following it.

The exploration problem is feasible in the sense that a universal traversal sequence always exists for fully regular graphs. To see this, follow any path in an edge-colored graph and then return to the starting location by backtracking along the same path to get a color sequence that is a palindrome. Conversely, following a color sequence that is a palindrome guarantees to yield a closed tour, irrespective of the graph and the starting location. This means that we can obtain a universal traversal sequence by chaining

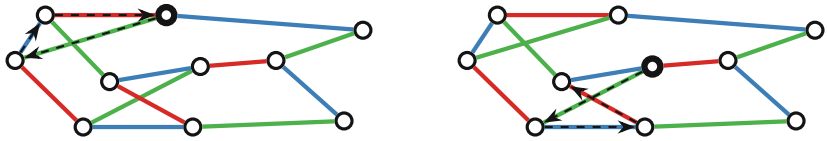

Fig. 1. A regular graph with two different starting locations. By following the color sequence "green, blue, red", the agent either moves on a cycle (left) or not (right), but there is no way to distinguish between these two cases as vertices are indistinguishable. (Color figure online)

together all color sequences that are palindromes in order of increasing lengths. The resulting sequence is guaranteed to follow every path from the starting location, irrespective of the graph, and thus to eventually visit all vertices.

For non-regular graphs, a universal traversal sequence seems unattainable since not every color needs to be available at every vertex. However, the exploration of an arbitrary non-regular graph $G = (V, E)$ can be reduced to the exploration of a fully regular graph $G_{\text{freg}} = (V_{\text{freg}}, E_{\text{freg}})$ via the construction shown in Fig. 2. To this end, we first construct a regular graph $G_{\text{reg}} = (V_{\text{reg}}, E_{\text{reg}})$ with bi-colored edges. For every vertex $v \in V$ and each color c of its adjacent edges, we introduce a color copy $(v, c) \in V_{\text{reg}}$, connect the color copies of v in a cycle and add the original edges between the respective color copies. The resulting graph has only three colors. The edges in the cycles are bi-colored with one color pointing to the next color copy, and one color pointing to the previous color copy. Edges between color copies of different vertices have a third color. We proceed to eliminate the bi-colored edges in G_{reg} and obtain a fully regular graph G_{freg}. This can be done by first adding an intermediate vertex for each bi-colored edge, and then mirroring (i.e.,copying) the entire construction and connecting each vertex of degree 2 with its reflection with the third color.

As explained above, there is a universal traversal sequence for 3-regular graphs and, thus, the sequence also explores G_{freg}. Given a universal traversal sequence for G_{freg}, we can explore G with an additional memory overhead that is logarithmic in the maximum degree of the original graph and, thus, in $\mathcal{O}(\log n)$. The idea is to perform a virtual traversal of G_{freg} and only actually move in G whenever the virtual traversal transitions between color copies of different vertices of G. The memory is used to store which color copy of its location in G the agent is (virtually) located at in G_{freg}, as well as whether it is at a vertex or its reflection and whether it is located on the intermediate vertex of a bi-colored edge.

While we have now established the general feasibility of the exploration problem, the constructed exploration algorithm is not very satisfactory in the sense that it enumerates an exponential number of sequences before all vertices are guaranteed to have been visited. This means that the algorithm requires an exponential number of moves and a linear memory size to keep track of its current state. Note that as long as the color sequence remains aperiodic, linear memory is needed to perform an exponential number of steps and, conversely, making use of a linear number of memory bits means visiting an exponential number of memory states and thus an exponential running time. In that sense, there is a direct correspondence between exponential time and linear memory. From now on, we focus on memory usage only. The natural question in this context becomes: Can we solve the exploration problem in sub-linear memory?

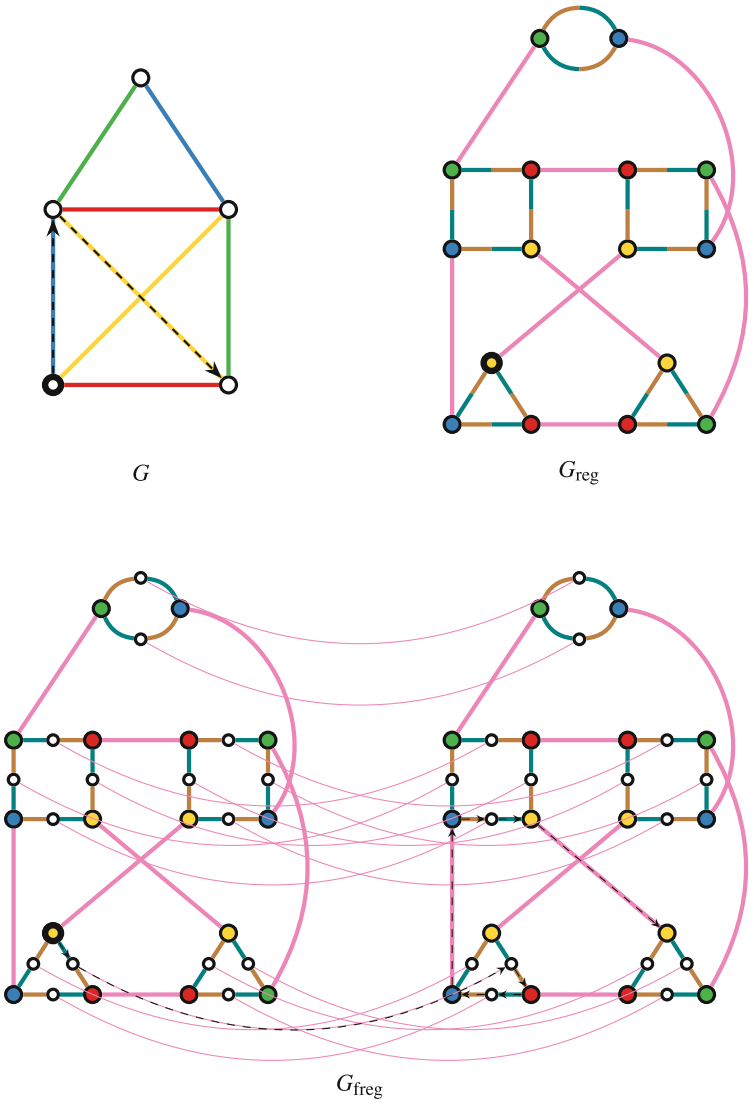

Fig. 2. Turning an arbitrary graph G into a regular graph G_{reg} with bi-colored edges and further into a fully regular graph G_{freg}. In the construction, we order the four colors of G cyclically as yellow-red-green-blue. In G_{reg}, brown edges point to the next color available at the corresponding vertex in G, teal edges point to the previous color, and purple edges move to a color copy of another vertex. To construct G_{freg} from G_{reg}, an intermediate vertex is added to the center of each bi-colored edge, the graph is copied, and two corresponding intermediate vertices are connected by a purple edge. Starting with the color yellow on the left copy in G_{freg}, the color sequence "teal-purple-brown-teal-brown-purple-brown-teal-purple" for G_{freg} leads to the movement along a blue edge and a yellow edge as indicated in G. (Color figure online)

3 Trapping a Single Agent

To approach the question of how much memory is necessary in general to deterministically explore a graph G of size n, we first need to realize how insufficient memory can manifest itself in terms of the inability of the agent to explore: Essentially, the only way that the agent may fail to explore G in finite time is by getting "trapped" in periodic behavior that forces it to move on a closed tour eternally, without having visited all vertices. With this in mind, we make the following definition.

Definition 1. *A* trap *for an exploration algorithm is given by an edge-colored graph G together with an initial location v_0, such that the algorithm never visits all vertices of G when starting at v_0.*

We fix a deterministic exploration algorithm $\mathscr{A}$ with a finite number $b \in \mathbb{N}$ of memory bits and construct a trap of some size n for this algorithm. The size of our trap then bounds the largest size of graphs that the algorithm can explore. Conversely, since the construction can be carried out for any deterministic algorithm, we obtain a lower bound on the required number of memory bits necessary to explore graphs of size (up to) n.

To construct a trap G for $\mathscr{A}$, first observe that $\mathscr{A}$ has at most 2^b different memory states at its disposal. Our construction ensures that G is a fully regular graph of degree 3, using a fixed set of three colors C. As observed in the previous section, $\mathscr{A}$ is sure to yield the same sequence S of colors for all fully regular graphs using colors C and irrespective of the initial location v_0. Since $\mathscr{A}$ has at most 2^b different states, it must enter at least one state for the second time within the first 2^b steps. Assume the same state is entered in steps $1 \leq i < j \leq 2^b$. Then the behavior of $\mathscr{A}$ and, consequently, S must become periodic after step i, i.e.,$S = (c_1, \dots, c_{i-1}) \oplus S_{\mathrm{p}}^{\infty}$, where '$\oplus$' denotes concatenation of sequences, and S_{p} is a finite subsequence of S of length $j-i$.

Consider the infinite walk $W = (v_0, v_1, v_2, \dots)$ induced by S in the infinite 3-regular tree where the set of colors of the edges incident to each vertex is C; cf. Fig. 3 (top). By definition, $\mathscr{A}$ is in the same memory state after steps i and j, implying that it follows the same infinite color sequence starting at v_i in steps $i+1, i+2, \dots$ as it does starting at v_j in steps $j+1, j+2, \dots$. Assume that $v_i = v_j$. Then, the algorithm moves on a closed tour of length $j-i$ after step i while having visited at most $i+j-i = j \leq 2^b$ different vertices. We can now take the subgraph G of the infinite tree induced by all edges incident to vertices in W as our trap. Note that this graph need not be fully regular, but we can add missing edges by mirroring G as before (cf. Sect. 2) and connecting corresponding vertex pairs of degree smaller three by an edge of a color they are missing. This decreases the number of missing colors at all vertices of degree smaller three and needs to be repeated at most once to make the graph fully regular.

In the case $v_i \neq v_j$ the algorithm may visit an infinite number of different vertices. The intuitive idea now is to "close a loop" by ensuring that both the edges of color $c_{i+1} = c_{j+1}$ at v_i and at v_j lead to the same vertex. Of course, we cannot simply replace the edge of color c_{j+1} at v_j by the edge $\{v_j, v_{i+1}\}$ of the same color, since we also need to keep the edge $\{v_i, v_{i+1}\}$ of this color. However, we can achieve the same result by "folding" v_i onto v_j, i.e.,by identifying $v_i = v_j$ and identifying the predecessors of v_i along W and their neighborhoods accordingly. More precisely, we identify each vertex

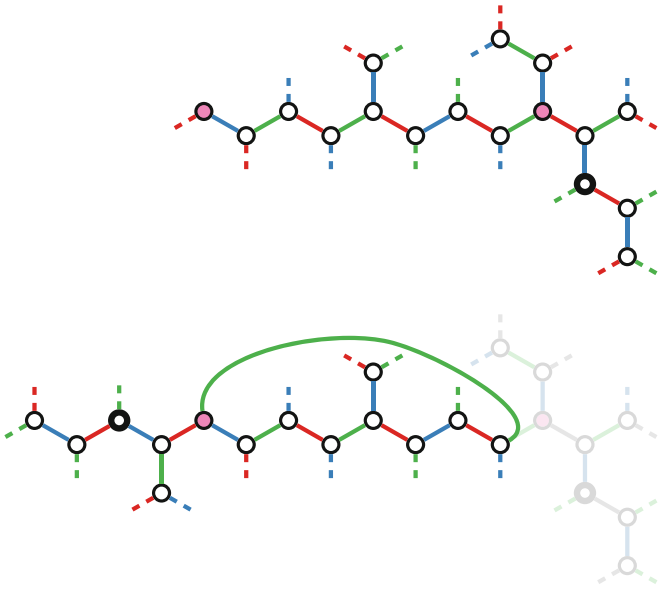

Fig. 3. Construction of a trap for a single agent with b bits of memory. Top: After at most 2^b steps in a fully regular graph, the same memory state must repeat (purple vertices). Bottom: Closing a loop to trap the agent on a closed walk. (Color figure online)

v adjacent to v_i with the unique vertex v' adjacent to v_j such that the colors of the edges $\{v_i, v\}$ and $\{v_j, v'\}$ coincide. We repeat this process for all vertices $v_{i-1}, \ldots, v_0$ along W; cf Fig. 3 (bottom). Afterwards, we again take the subgraph induced by $\{v_0, \ldots, v_j\}$ together with their neighbors as our trap, making it fully regular as before.

In either case, we have constructed a trap of size $n = \mathcal{O}(2^b)$. Since we can perform this construction for any deterministic algorithm with b memory bits, this implies a lower bound of $\Omega(\log n)$ on the required number of memory bits to explore every graph of size up to $n \in \mathbb{N}$. We have shown the following.

Theorem 1 (Fraigniaud et al. [12]). *The number of memory bits needed for undirected, deterministic graph exploration is* $\Omega(\log n)$.

4 Reingold's Algorithm

We will see that the lower bound shown in Sect. 3 on the memory needed to explore an undirected graph deterministically is tight, i.e.,undirected graphs with n vertices can be explored deterministically with $\mathcal{O}(\log n)$ memory. This algorithmic result follows from a famous result of Reingold [16] in which he established that USTCON $\in \mathscr{L}$. Here, $\mathscr{L}$ is the class of problems solvable with logarithmic memory and USTCON is the problem of deciding, for a given undirected graph $G = (V, E)$ and two designated vertices $s, t \in V$, whether s and t are connected in G. The algorithm devised by Reingold for his proof can be turned into a log-space exploration algorithm, which we outline in the following.

We first argue that fully regular graphs with constant degree and good vertex expansion can be explored with logarithmic memory. Suppose the graph G is fully regular with constant degree d and enjoys the property that there is a constant $\varepsilon > 0$ such that for all vertex sets $S \subset V$ with $|S| \leq n/2$ there are at least $(1+\varepsilon)|S|$ vertices that are connected by an edge to a vertex in S. An upshot of this vertex expansion property is that the graph has at most logarithmic diameter. Indeed, for an arbitrary vertex $u \in V$ there are more than $n/2$ vertices within a distance of $k = \frac{\log(n/2)}{\log(1+\varepsilon)} + 1$ of u, so that every pair of vertices has a common vertex within distance k and, thus, the diameter is at most $2k \in \mathcal{O}(\log n)$. Similar to the argument in Sect. 2, it suffices to enumerate all returning color sequences of length $2k$ which can be done with $\mathcal{O}(\log n)$ space.

Regularity can be achieved with the transformation from G to G_{reg} explained in Sect. 2. Here, we stick to G_{reg} with its bi-colored edges instead of transforming G_{reg} further into G_{freg} since the bi-colored edges of G_{reg} do not harm our further arguments. We proceed to describe further transformations that turn G_{reg} into another regular graph G_{exp} with good vertex expansion. Let G be a fully d-regular graph with n vertices and let H be a c-regular graph with d vertices where c and d are constants. Then the *replacement product* $G \textcircled{r} H$ is the graph where each vertex v in G is replaced by a copy of H that we call the *cloud* of v. The edges within a cloud keep the colors that they have in H. For each edge of G, we introduce an edge with a new inter-cloud color between the respective vertices in the corresponding clouds; cf. Fig. 4. The resulting graph $G \textcircled{r} H$ is fully regular with degree $c+1$. Based on the replacement product $G \textcircled{r} H$, we introduce another graph product, the *zig-zag product* $G \textcircled{z} H$. The zig-zag-product $G \textcircled{z} H$ has the same set of vertices as the replacement product $G \textcircled{r} H$, but only edges between vertex clouds of different vertices. Specifically, let (u,i) be a vertex belonging to the cloud of u, and (v,j) be a vertex belonging to the cloud of v. Then, the edge $\{(u,i),(v,j)\}$ is contained in the replacement product if and only if there is path of length three from (u,i) to (v,j) in $G \textcircled{r} H$ where the middle edge is an edge between different clouds. For a vertex (u,i) there are exactly c^2 such paths starting in (u,i): the first degree of freedom is to choose one of c colors (within the current cloud), then change the cloud with an inter-cloud edge, and then choose one of c colors for the second cloud. Associating each of these c^2 color combinations with a new color in $G \textcircled{z} H$, we obtain that $G \textcircled{z} H$ is fully regular with degree c^2. We note that this construction also works if G has bi-colored edges by allowing inter-cloud edges also between different copies of vertices of H. In any case, we may end up with a graph $G \textcircled{z} H$ having bi-colored edges. Suppose that H is of constant size and that we have a traversal sequence for $G \textcircled{z} H$. Then, every edge traversal in $G \textcircled{z} H$ corresponds to three edge traversals in $G \textcircled{r} H$. We maintain a stack of future edge traversals in $G \textcircled{r} H$. Since H has constant degree, so has $G \textcircled{r} H$, and we can store this stack with up to three colors in constant memory. In this way, we obtain a traversal sequence for $G \textcircled{r} H$ with constant memory overhead. From a traversal sequence for $G \textcircled{r} H$, we further obtain a traversal sequence for G by memorizing the current copy of the vertex of H within the current cloud similar to the virtual traversal of G_{freg} in Sect. 2. As H has constant size, this requires only constant memory overhead. We conclude that a traversal sequence for $G \textcircled{z} H$ can be used to traverse G with constant memory overhead.

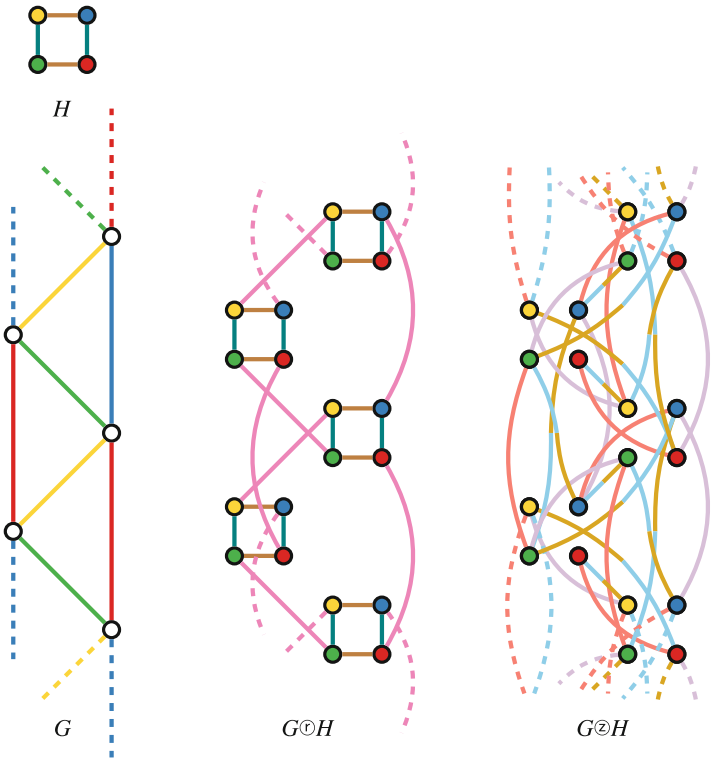

Fig. 4. The replacement product $G ⓡ H$ and the zig-zag-product $G ⓩ H$ for two graphs G and H. In the replacement product $G ⓡ H$, edges within a cloud keep the colors they had in H, here brown or teal. The edges between clouds get a new inter-cloud color, here purple. Every edge in $G ⓩ H$ corresponds to a path of length three in $G ⓡ H$ where the middle edge is purple, e.g.,an edge color gold in $G ⓩ H$ corresponds to a path in $G ⓡ H$ that is teal-pink-brown. (Color figure online)

It is left to show that we can transform G_{reg} into a graph with good vertex expansion. In order to show that a d-regular graph has good vertex expansion, it suffices to show that the second largest eigenvalue λ of the normalized adjacency matrix is bounded from above by a constant strictly smaller than 1; cf. Tanner [21], Alon and Milnan [3], and Alon [2]. For the normalized adjacency matrix $M = (m_{u,v})_{u,v \in V}$, the entry $m_{u,v}$ is defined as $1/d$ times the number of edges from u to v. For ease of notation, we call a d-regular graph on n vertices an (n,d,α)-graph if $\lambda \leq \alpha$. We use the following properties of the second largest eigenvalues of regular graphs:

1. Alon and Sudakov [5]:
   A d-regular, connected, non-bipartite n-vertex graph is a $\left(n,d,1-\frac{1}{dn^2}\right)$-graph.
2. Basic linear algebra:
   Taking the k-th power of a graph means introducing an edge for each k-edge path in the original graph. If G is an (n,d,λ)-graph, then its k-th power is an (n,d^k,λ^k)-graph.

3. Alon and Roichman [4] (cf.discussion in Reingold et al. [17, § 5]):
   There exists a $(c^{16}, c, 1/2)$-graph for some constant c.
4. Reingold et al. [17]:
   Let G be an (n,d,λ)-graph and let H be a $(d,c,1/2)$-graph. Then $G ⓩ H$ is an $\left(nd, c^2, \frac{1}{8}(3\lambda + \sqrt{9\lambda^2+16})\right)$-graph.

Let H be a $(c^{16}, c, 1/2)$-graph with c constant as in Property 3.. For an arbitrary graph G on n vertices, first construct G_{reg}. Let G_0 be equal to G_{reg} except that $c^{16}-3$ self loops are added to each vertex. Let $\ell = 2\lceil \log(c^{16}n^4)\rceil$. For $i = 1,\dots,\ell$, define $G_i = (G_{i-1} ⓩ H)^8$, i.e.,to obtain the next graph in the sequence, we first apply the zig-zag product with H and then take the 8-th power of the resulting graph. Note that this is well-defined since $G_{i-1} ⓩ H$ has degree c^2, so that $G_i = (G_{i-1} ⓩ H)^8$ has degree c^{16}, and $G_i ⓩ H$ is defined. Any traversal sequence for G_i can be transformed with constant memory overhead to a traversal sequence for G_{i-1}, since it involves taking the zig-zag product with a graph of constant size and power 8 (which requires to memorize up to 7 additional steps). Thus, a traversal sequence for G_ℓ can be transformed to a traversal sequence for G_0 and, hence, an exploration sequence for G with memory overhead of $\mathcal{O}(\ell) = \mathcal{O}(\log n)$. It remains to show that G_ℓ has good vertex expansion. We claim that $\lambda(G_i) \le \max\{\lambda(G_{i-1})^2, 1/2\}$ for all $i = 1,\dots,\ell$. To prove the claim, let $\lambda = \lambda(G_{i-1})$ and note that, by Property 4.,

$$\lambda(G_{i-1} ⓩ H) \le \frac{1}{8}\left(3\lambda + \sqrt{9\lambda^2+16}\right) \le \frac{1}{8}\left(3\lambda + 5\right) = 1 - \frac{3}{8}\left(1-\lambda\right) < 1 - \frac{1}{3}\left(1-\lambda\right),$$

implying $\lambda(G_i) < \left(1 - \frac{1}{3}(1-\lambda)\right)^8$ by Property 2. If $\lambda < \frac{1}{2}$, then $\lambda(G_i) < \left(\frac{5}{6}\right)^8 < \frac{1}{2}$. Otherwise, it is straightforward to verify that the function $f(x) = (1 - \frac{1}{3}(1-x))^4$ is convex on $[0,1]$ and $1 \ge f(1)$ as well as $1/2 \ge f(1/2)$. We conclude that $f(x) \le x$ for all $x \in [1/2, 1]$, in particular

$$\left(1 - \frac{1}{3}\left(1-\lambda\right)\right)^4 \le \lambda,$$

implying $\lambda(G_i) \le \lambda^2$. Finally, the graph G_0 is regular with degree c^{16} and has at most n^2 nodes. By Property 1. this implies that $\lambda(G_0) \le 1 - \frac{1}{c^{16}n^4}$. With the claim above, we obtain

$$\lambda(G_\ell) \le \max\left\{\left(1 - \frac{1}{c^{16}n^4}\right)^{2^\ell}, \frac{1}{2}\right\} \le \left\{\left(\left(1 - \frac{1}{c^{16}n^4}\right)^{c^{16}n^4}\right)^2, \frac{1}{2}\right\} \le \left\{\left(1 - \frac{1}{e}\right)^2, \frac{1}{2}\right\} = \frac{1}{2}.$$

As we sketched above, the transformation from G to G_ℓ requires only logarithmic memory and can be conducted locally, i.e.,a traversal sequence for G_ℓ can be transformed into an exploration sequence for G with logarithmic space overhead. Finally,

we eliminate the bi-colored edges of G_ℓ as in Sect. 2. Since this construction increases the diameter of the graph by at most a factor of 2, it has still a logarithmic diameter, so that a traversal sequence can be constructed with logarithmic space. This yields the following result.

Theorem 2 (Reingold [16]). *Undirected graphs can be deterministically explored with an agent that has $\mathcal{O}(\log n)$ bits of memory.*

5 Trapping Multiple Agents

After having established that $\Theta(\log n)$ memory bits are necessary and sufficient for deterministic exploration with a single agent, we now investigate whether this bound can substantially be lowered by allowing additional agents. More precisely, we consider a setting with $k \geq 2$ deterministic and distinguishable agents that behave as before individually, but move in a synchronized fashion and may exchange information while co-located at a vertex. To see that allowing collaboration makes a fundamental difference, even for $k = 2$, observe that, for example, two agents can distinguish closed tours simply by leaving one of them at the starting location; cf. Fig. 1. This additional power is also evidenced by a drastically increased difficulty of constructing traps: For a long time, the smallest known traps for k agents, with s memory states each, had a size of $\mathcal{O}\left(s^{s^{\cdot^{\cdot^{s}}}}\right)$ with $\mathcal{O}(k)$ levels in the exponent (Fraigniaud et al. [13], Rollik [18]), compared to the singly exponential bound of Theorem 1.

To see that a substantially different approach is needed to trap multiple agents, recall the construction in Sect. 3: The intuitive idea was to add vertices along a tree until the agent enters a memory state for the second time, at which point we close a loop. Since the number of memory states available to the agent is bounded by a constant, namely 2^b, this yielded a trap of singly exponential size in b. The key difference when allowing multiple agents is that the behavior of the agents no longer only depends on their collective memory state. It now might make a difference in the behavior of the algorithm at what points agents meet – which is exactly the reason, why they can distinguish cycles, as explained above. This means that the behavior of the algorithm may depend in a non-trivial way on the positions of the agents in the graph, relative to each other. As we increase the number of vertices n, the number of such configurations grows as n^k, and we can no longer hope for configurations to ever repeat.

The key idea to overcome this is to force the agents to stay "close" to each other, which ensures that the number of configurations stays bounded and allows us to use the same general approach as before. The following informal definition generalizes the notion of a trap to multiple agents.

Definition 2. *A k-*barrier *B_k in a graph G for an algorithm $\mathscr{A}$ is a subgraph of G whose removal disconnects the graph into two connected components, with the property that no agent ever traverses B_k from one component to the other without at least k other agents entering B_k during the traversal.*

In particular, a 1-barrier plays the role of a simultaneous trap for every individual agent. Note that agents may behave differently from one another, so we need to deal with each

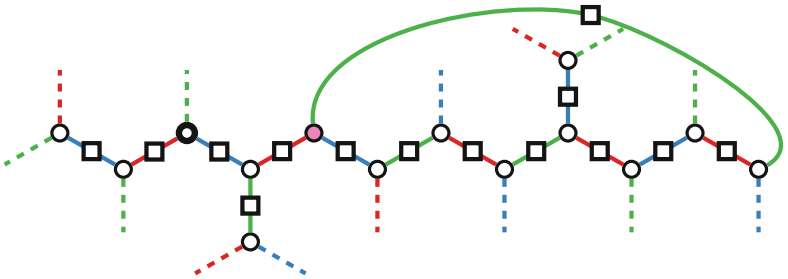

Fig. 5. Sketch of the construction of an i-barrier. Boxes indicate $(i-1)$-barriers.

one using a separate construction. We have seen in Sect. 3 how to construct a trap for a single agent, and we can essentially chain traps together for the individual agents in order to obtain a 1-barrier. We will now describe how to recursively construct i-barriers for $i \in \{2,\dots,k\}$. Once we have constructed a k-barrier, we have the desired trap for the set of all k-agents.

The idea of the recursive construction of an i-barrier is to use the same approach as in the trap for a single agent, but replacing every edge by an $(i-1)$-barrier; cf. Fig. 5. More precisely, we fix any set of i agents and assume that only these agents enter our construction. Since, on a meta-level, edges are now $(i-1)$-barriers, the agents can only traverse these "meta-edges" if they all enter the corresponding barrier, i.e.,if they stay somewhat close together. Essentially, throughout the traversal, all agents are guaranteed to be located in one of the three $(i-1)$-barriers surrounding some meta-vertex. Of course, the same is true recursively within every $(i-1)$-barrier containing at most $i-1$ of the agents. By a careful recursive inspection, the total number of configurations of the agents can be bounded independently of the number of meta vertices. This allows a similar approach as before: Add meta-vertices until a configuration repeats and close a loop to obtain a trap. To obtain an i-barrier, we again need to chain traps for every subset of i agents together.

With some refinement and a thorough analysis, it can be shown that this yields a k-barrier, and thus a trap, of size $\mathcal{O}(s^{2^{5k}})$ for k agents with s memory states each. In other words, the agents can explore graphs of size up to $n \leq s^{2^{5k}}$, i.e.,$\log n \leq 2^{5k} \cdot \log s$ has to hold. Assuming that each agent has $\mathcal{O}(\log^{1-\varepsilon} n)$ bits of memory for some $\varepsilon \in (0,1)$, i.e.,just shy of the number needed to explore the graph on its own, we obtain $\log s = \mathcal{O}(\log^{1-\varepsilon} n)$. Combining both bounds and taking logarithms yields $k = \Omega\left(\log\left(\frac{\log n}{\log^{1-\varepsilon} n}\right)\right) = \Omega(\log\log n)$. This means that we need at least $k = \Omega(\log\log n)$ agents to explore undirected graphs of size n, even if every agent has almost enough memory to explore on its own!

Theorem 3. (Disser et al. [10 SPP]). *Deterministic exploration of undirected graphs needs at least $\Omega(\log\log n)$ agents if we allow $\mathcal{O}(\log^{1-\varepsilon} n)$ bits of memory for every agent, where $\varepsilon > 0$.*

6 Multi-agent Exploration

We outline the design of a collaborative exploration algorithm that matches the lower bound of Theorem 3, i.e.,we show that $\mathcal{O}(\log\log n)$ agents with sub-logarithmic memory are sufficient to explore unknown graphs of size up to n. Observe that $\mathcal{O}(\log n)$ agents are trivially sufficient by Reingold's algorithm (Theorem 2), since we can let agents move together and make each one responsible for maintaining a constant number of memory bits.

We start with a single agent with a constant number $m_0 \in \mathbb{N}$ of memory bits and show how to iteratively boost its memory by using a small number of additional agents. First consider how much progress, in terms of visiting vertices, the agent is able to accomplish on its own. For a single agent, we already know Reingold's algorithm which needs logarithmic space. Expressed differently, executing Reingold's algorithm with m_0 bits of available memory guarantees that the agent visits a number of distinct vertices of order $\Omega(2^{m_0})$, or completes the exploration.

These vertices can be visited multiple times, and, in general, there is no way of knowing the order in which the vertices appear during the traversal T produced by Reingold's algorithm. However, using one additional agent to mark vertices and multiple repetitions of traversal with Reingold's algorithms for different positions of the additional agent, it can be shown that we can treat T as a simple cycle without self intersections. Assuming that the agent has this cycle T of length $\Omega(2^{m_0})$, for some constant $c \in \mathbb{N}$, that it can navigate systematically, it can position a constant number $a \in \mathbb{N}$ of additional agents along T. Since agents are distinguishable, there are $|T|^a$ configurations that can be established in this way. The key idea now is to use the configuration of the agents along T as a form of virtual memory state, in order to boost the amount of memory available to the agent.

The number of memory bits that can be encoded in this way is $m_1 = \log|T|^a$, which is of order am_0. This means that we have boosted the initial memory capacity roughly by a factor of a. Having more (virtual) memory at its disposal, the agent can now recursively repeat the procedure, again boosting the memory by another factor of a, and so on. After $\log\log n$ levels of recursion, the amount of virtual memory is of order $a^{\log\log n} \cdot m_0 = \Omega(\log n)$. But we already know that this is sufficient to complete the exploration, by Theroem 2.

For this approach to yield the claimed bound, it is crucial to argue that only a constant number of agents and memory bits are needed in each recursive level, not only to encode, but also to manipulate the virtual memory. In particular, in each move performed in some level of the recursion, the agents encoding the virtual memory on lower recursive levels need to be moved in the graph to stay in the same positions relative to the agent. It can be shown that this is indeed possible with a constant overhead in agents, and we obtain the following tight result.

Theorem 4. (Disser et al. [10 SPP]). *Undirected graphs can be deterministically explored with $\mathcal{O}(\log\log n)$ agents, even if we only allow constant memory for every agent.*

7 Bibliographic Notes

The first exploration algorithms were designed for mazes. A *maze* is a subgraph of the two-dimensional grid where the vertices are indistinguishable and each edge is labeled with its cardinal direction. To facilitative the exploration, the agent is sometimes equipped with a set of distinguishable pebbles that can be dropped and retrieved at nodes. After initial non-tight results (Blum and Sakoda [7], Budach [8], Shah [20]), it has been proven that an agent with finite memory needs two pebbles to explore any maze (Blum and Kozen [6]) and that one pebble does not suffice (Hoffmann [14]). Blum and Kozen [6] further showed that also two agents with finite memory can explore all mazes.

General undirected graphs are harder to explore. The lower bound of $\Theta(\log n)$ on the memory needed by a single agent to explore all undirected vertex graphs deterministically given in Sect. 3 is due to Fraigniaud et al. [12]. Aleliunas et al. [1] showed that a random walk of length $n^5 \log n$ explores an undirected n-vertex graph with high probability. The deterministic algorithm exploring undirected vertex graphs explained in Sect. 4 is due to Reingold [16]. We here follow the presentation of the algorithm and the analysis of Reingold's original paper. There are also alternative proofs for this result that avoid the use of the zig-zag-product; see Rozenman and Vadhan [19]. Reingold's algorithm constructs a universal exploration sequence. This concept was introduced by Koucky [15].

Regarding the exploration of a graph by a set of cooperating agents, Blum and Kozen [6] showed that three agents with finite memory cannot explore all finite undirected planar graphs. Rollik [18] strengthened this result showing that for any number $k \in \mathbb{N}$ of agents with $s \in \mathbb{N}$ states, there is a *trap* of size $\mathcal{O}\left(s^{s^{\cdot^{\cdot}}}\right)$ with $2k+1$ levels in the exponent, i.e.,a graph that the agents cannot explore. Fraigniaud et al. [13] improved this bound to $k+1$ levels in the exponent. The non-planar trap of size $\mathcal{O}\left(s^{2^{5k}}\right)$ given in Sect. 5 is due to Disser et al. [10 SPP]. This result implies that if each agent has a sublogarithmic memory of $\mathcal{O}(\log^{1-\varepsilon} n)$ with $\varepsilon > 0$, then $\mathcal{O}(\log\log n)$ agents are needed to explore all undirected n-vertex graphs. Another consequence from their construction is that a single agent with sublogarithmic memory needs $\mathcal{O}(\log\log n)$ pebbles to explore all undirected n-vertex graphs. The result that $\mathcal{O}(\log\log n)$ agents with constant memory can explore all undirected n-vertex graphs presented Sect. 6 is due to Disser et al. [10 SPP]. They actually showed that a single agent with constant memory and $\mathcal{O}(\log\log n)$ pebbles can explore the graph and provide a general reduction from agents to pebbles. They further proved that their algorithm runs in polynomial time. For results regarding the exploration time needed by an agent with unconstrained memory, see Dudek et al. [11] and Chalopin et al. [9].

References

1. Aleliunas, R., Karp, R.M., Lipton, R.J., Lovász, L., Rackoff, C.: Random walks, universal traversal sequences, and the complexity of maze problems. In: FOCS, pp. 218–223. IEEE Computer Society (1979). https://doi.org/10.1109/SFCS.1979.34
2. Alon, N.: Eigenvalues and expanders. Combinatorica **6**(2), 83–96 (1986). https://doi.org/10.1007/BF02579166
3. Alon, N., Milman, V.D.: λ_1, isoperimetric inequalities for graphs, and superconcentrators. J. Comb. Theory, Ser. B **38**(1), 73–88 (1985). https://doi.org/10.1016/0095-8956(85)90092-9

4. Alon, N., Roichman, Y.: Random Cayley graphs and expanders. Random Struct. Algorithms **5**(2), 271–285 (1994). https://doi.org/10.1002/rsa.3240050203
5. Alon, N., Sudakov, B.: Bipartite subgraphs and the smallest eigenvalue. Comb. Probab. Comput. **9**(1), 1–12 (2000). https://doi.org/10.1017/S0963548399004071
6. Blum, M., Kozen, D.: On the power of the compass (or, why mazes are easier to search than graphs). In: FOCS, pp. 132–142. IEEE Computer Society (1978). https://doi.org/10.1109/SFCS.1978.30
7. Blum, M., Sakoda, W.J.: On the capability of finite automata in 2 and 3 dimensional space. In: FOCS, pp. 147–161. IEEE Computer Society (1977). https://doi.org/10.1109/SFCS.1977.20
8. Budach, L.: Automata and labyrinths. Math. Nachrichten **86**, 195–282 (1978). https://doi.org/10.1002/mana.19780860120
9. Chalopin, J., Das, S., Kosowski, A.: Constructing a map of an anonymous graph: applications of universal sequences. In: Lu, C., Masuzawa, T., Mosbah, M. (eds.) OPODIS 2010. LNCS, vol. 6490, pp. 119–134. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17653-1_10
10 SPP. Disser, Y., Hackfeld, J., Klimm, M.: Tight bounds for undirected graph exploration with pebbles and multiple agents. J. ACM **66**(6), 40:1–40:41 (2019). https://doi.org/10.1145/3356883
11. Dudek, G., Jenkin, M., Milios, E.E., Wilkes, D.: Robotic exploration as graph construction. IEEE Trans. Robot. Autom. **7**(6), 859–865 (1991). https://doi.org/10.1109/70.105395
12. Fraigniaud, P., Ilcinkas, D., Peer, G., Pelc, A., Peleg, D.: Graph exploration by a finite automaton. Theor. Comput. Sci. **345**(2–3), 331–344 (2005). https://doi.org/10.1016/j.tcs.2005.07.014
13. Fraigniaud, P., Ilcinkas, D., Rajsbaum, S., Tixeuil, S.: The reduced automata technique for graph exploration space lower bounds. In: Goldreich, O., Rosenberg, A.L., Selman, A.L. (eds.) Theoretical Computer Science. LNCS, vol. 3895, pp. 1–26. Springer, Heidelberg (2006). https://doi.org/10.1007/11685654_1
14. Hoffmann, F.: One pebble does not suffice to search plane labyrinths. In: Gécseg, F. (ed.) FCT 1981. LNCS, vol. 117, pp. 433–444. Springer, Heidelberg (1981). https://doi.org/10.1007/3-540-10854-8_47
15. Koucký, M.: Universal traversal sequences with backtracking. J. Comput. Syst. Sci. **65**(4), 717–726 (2002). https://doi.org/10.1016/S0022-0000(02)00023-5
16. Reingold, O.: Undirected connectivity in log-space. J. ACM **55**(4), 17:1–17:24 (2008). https://doi.org/10.1145/1391289.1391291
17. Reingold, O., Vadhan, S., Wigderson, A.: Entropy waves, the zig-zag graph product, and new constant-degree expanders. Ann. Math. **155**(1), 157–187 (2002). https://doi.org/10.2307/3062153
18. Rollik, H.: Automaten in planaren graphen. Acta Informatica **13**, 287–298 (1980). https://doi.org/10.1007/BF00288647
19. Rozenman, E., Vadhan, S.: Derandomized squaring of graphs. In: Chekuri, C., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.) APPROX/RANDOM -2005. LNCS, vol. 3624, pp. 436–447. Springer, Heidelberg (2005). https://doi.org/10.1007/11538462_37
20. Shah, A.N.: Pebble automata on arrays. Comput. Graph. Image Process. **3**(3), 236–246 (1974). https://doi.org/10.1016/0146-664X(74)90017-3
21. Tanner, R.M.: Explicit concentrators from generalized n-gons. SIAM J. Alg. Disc. Meth. **5**(3), 287–293 (1984). https://doi.org/10.1137/0605030