Brief Announcement: Self-Stabilizing Graph Exploration by a Single Agent

Yuichi Sudo
Hosei University, Tokyo, Japan

Fukuhito Ooshita
Fukui University of Technology, Fukui, Japan

Sayaka Kamei
Hiroshima University, Hiroshima, Japan

Abstract

In this paper, we present two self-stabilizing algorithms that enable a single (mobile) agent to explore graphs. The agent visits all nodes starting from any configuration, *i.e.,* regardless of the initial state of the agent, the initial states of all nodes, and the initial location of the agent. We evaluate the algorithms using two metrics: cover time, which is the number of moves required to visit all nodes, and memory usage, which includes the storage needed for the state of the agent and the state of each node. The first algorithm is randomized. Given an integer $c = \Omega(n)$, the cover time of this algorithm is optimal, *i.e.,* $O(m)$ in expectation, and the memory requirements for the agent and each node v are $O(\log c)$ and $O(\log(c + \delta_v))$ bits, respectively, where n and m are the numbers of nodes and edges, respectively, and δ_v is the degree of v. The second algorithm is deterministic. It requires an input integer $k \geq \max(D, \delta_{\max})$, where D and $\delta_{\max}$ are the diameter and the maximum degree of the graph, respectively. The cover time of this algorithm is $O(m + nD)$, and it uses $O(\log k)$ bits both for agent memory and each node.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases mobile agents, self-stabilization, graph exploration

Digital Object Identifier 10.4230/LIPIcs.DISC.2024.55

Related Version *Full Version*: `https://doi.org/10.48550/arXiv.2010.08929` [16]

Funding *Yuichi Sudo*: JSPS KAKENHI 20H04140 and 20KK0232, and JST FOREST Program JPMJFR226U.
Fukuhito Ooshita: JSPS KAKENHI 22K11903.
Sayaka Kamei: JSPS KAKENHI 23K11059 and 23K28037.

1 Introduction

We focus on the *exploration problem* involving a single mobile entity, referred to as a mobile agent or simply an *agent*, within any undirected, simple, and connected graph $G = (V, E)$. This agent, functioning as a finite state machine, migrates from node to node via edges at each time step. Upon visiting a node, the agent can access and modify the node's local memory, known as a *whiteboard.* The graph is anonymous, *i.e.,* nodes lack unique identifiers. Our objective is to enable the agent to visit every node in the graph in as few steps as possible while minimizing the memory usage of both the agent and the whiteboards. This exploration problem, fundamental in the study of mobile computing entities, has been extensively studied [13, 11, 18, 7, 8, 15]. Exploration algorithms have frequently served as a foundation for solving other fundamental problems such as rendezvous, gathering, dispersion, and gossip sharing.

In this paper, we tackle the exploration problem under a more challenging setting: *self-stabilizing exploration* [13, 8]. We do not presuppose any specific initial global state (or *configuration*) of the network. This means that at the start of the exploration, (i) the agent's location within G is arbitrary, (ii) the agent's state is arbitrary, and (iii) the content of each whiteboard is arbitrary. The agent is required to visit all nodes in G from any potentially inconsistent configuration. Generally, an algorithm is considered self-stabilizing [4] for problem P if it can solve P starting from any configuration. Self-stabilizing algorithms are capable of handling any type of transient faults, such as temporary memory corruption, making their design both practically and theoretically significant.

Generally speaking, several studies tackle a variety of problems involving mobile agents in the self-stabilizing setting [2, 8, 10]. In this setting, the number of agents in the graph is fixed. In our case (*i.e.,* self-stabilizing exploration by a single agent), the number of agents is always exactly one: we do not consider configurations where no agent exists, or where two or more agents are present. Therefore, this setting may be particularly suitable for applications where physical robots operates in a field represented by an undirected graph, and the robots can leave information in some way at each intersection in the field.

One might think that this problem, self-stabilizing exploration by a single agent, fall outside the scope of distributed computing because only a single mobile agent is considered. However, we believe this is not the case, as the information accessible to the single agent is distributed throughout the entire graph. When minimizing agent memory, the agent must manage the necessary information distributed across the whiteboards throughout the graph. This situation often illustrates the trade-off between time complexity and agent memory, which is one of the essential aspects of distributed computing. Moreover, as mentioned earlier, exploration algorithms often serve as a fundamental building block for addressing other problems related to mobile agents. Therefore, our randomized and deterministic algorithms, introduced in this paper, could be used to solve various (more distributed) problems, such as rendezvous, gathering, gossiping, and leader election, in a self-stabilizing manner.

Throughout this paper, we denote the number of nodes, the number of edges, the diameter of a graph by n, m, and D, respectively. We denote the degree of a node v by δ_v, and define $\delta_{\min} = \min_{v \in V} \delta_v$ and $\delta_{\max} = \max_{v \in V} \delta_v$.

1.1 Related Work

If we are allowed to use randomization, we can easily solve the self-stabilizing exploration with a well known strategy called *the simple random walk.* When the agent visits a node $v \in V$, it simply chooses a node as the next destination uniformly at random among $N(v)$, where $N(v)$ is the set of all neighbors of v in G. In other words, it moves to any node $u \in N(v)$ with probability $P_{v,u} = 1/\delta_v$. It is well known that the agent running this simple algorithm visits all nodes in G within $O(\min(mn, mD \log n))$ steps in expectation where $n = |V|$, $m = |E|$, and D is the diameter of G. (See [1, 9].) Since the agent is oblivious (*i.e.,* the agent does not bring any information at a migration between two nodes) and does not use whiteboards, the simple random walk is obviously a self-stabilizing exploration algorithm.

Ikeda, Kubo, and Yamashita [7] improved the cover time (*i.e.,* the number of steps to visit all nodes) of the simple random walk by setting the transition probability as $P'_{v,u} = \delta_u^{-1/2} / \sum_{w \in N(v)} \delta_w^{-1/2}$ for any $u \in N(v)$. They proved that the cover time of this *biased random walk* is $O(n^2 \log n)$ steps in expectation. However, we cannot use this result directly in our setting because the agent must know the degrees of all neighbors of the current node to compute the next destination. We can implement this random walk, for example, as follows: every time the agent visits node v, it first obtains $(\delta_u)_{u \in N(v)}$ by visiting all v's neighbors in $2\delta_v$ steps, and then decides the next destination according to probability $(P'_{v,u})_{u \in N(v)}$, which is now computable with $(\delta_u)_{u \in N(v)}$. However, this implementation increases the cover time by a factor of at least $\delta_{\min}$ and at most $\delta_{\max}$. Whereas $n^2\delta_{\max} \log n > mn$ always holds, $n^2\delta_{\min} \log n < \min(mn, mD \log n)$ may also hold. Thus, we cannot determine which random walk has smaller cover time without detailed analysis. To bound the space complexity, we must know an upper bound Δ on $\delta_{\max}$ to implement this random walk. If the agent stores $(\delta_u)_{u \in N(v)}$ on v's whiteboard, it uses $O(\log \Delta)$ bits in the agent-memory and $O(\delta_v \log \Delta)$ bits in the whiteboard of each node v. If the agent stores $(\delta_u)_{u \in N(v)}$ only on the agent-memory, it uses $O(\Delta \log \Delta)$ bits in the agent-memory.

The algorithm given by Priezzhev, Dhar, Dhar, and Krishnamurthy [13], which is nowadays well known as the *rotor-router*, solves the self-stabilizing exploration deterministically. The agent is oblivious, but it uses only $O(\log \delta_v)$ bits in the whiteboard of each node $v \in V$. The edges $(\{v, u\})_{u \in N(v)}$ are assumed to be locally labeled by $0, 1, \dots, \delta_v - 1$ in a node v. The whiteboard of each node v has one variable $v.\mathtt{last} \in \{0, 1, \dots, \delta_v - 1\}$. Every time the agent visits a node v, it increases $v.\mathtt{last}$ by one modulo δ_v and moves to the next node via the edge labeled by the updated value of $v.\mathtt{last}$. This simple algorithm guarantees that starting from any configuration, the agent visits all nodes within $O(mD)$ steps [18]. Masuzawa and Tixeuil [8] also gave a deterministic self-stabilizing exploration algorithm. This algorithm itself is designed to solve the gossiping problem where two or more agents have to share their given information with each other. However, this algorithm has a mechanism to visit all the nodes starting from any configuration, which can be seen as a self-stabilizing exploration algorithm. The cover time and the space complexity for the whiteboards of this algorithm are asymptotically the same as those of the rotor-router, while it uses a constant space of the agent-memory, unlike oblivious algorithms such as the rotor-router.

In his seminal paper, Reingold [14] proved that given positive integer N, a Universal Exploration Sequence (UXS) with length *poly*(N) for (possibly non-simple) connected d-regular graphs with a size of at most N can be explicitly constructed in log-space and, hence, in polynomial time. Although we omit the definition of UXS here, from this result, we can immediately derive a self-stabilizing exploration algorithm for arbitrary graphs whose size is at most N, whose cover time is polynomial in N, with memory requirements of $O(\log N)$ bits for the agent and zero for the whiteboards. One might think that Reingold's UXS was designed for regular graphs, thus questioning its applicability to arbitrary graphs. However, this difference is not significant because we can virtually transform any arbitrary graph into a regular graph by adding self-loops (see [17] for details). Later, Ta-shma and Zwick [17] introduced the concept of a Strongly Universal Exploration Sequence (SUXS) and obtained results similar to those of Reingold, which allow us to improve the cover time of the above-mentioned self-stabilizing exploration algorithm from *poly*(N) to *poly*(n). Thus, the cover time no longer depends on a given upper bound N but only on the actual size n.

A few self-stabilizing algorithms were given for mobile agents to solve problems other than exploration. Blin, Potop-Butucaru, and Tixeuil [2] studied the self-stabilizing naming and leader election problem. Masuzawa and Tixeuil [8] gave a self-stabilizing gossiping algorithm. Ooshita, Datta, and Masuzawa [10] gave self-stabilizing rendezvous algorithms.

If we assume a specific initial configuration, that is, if we do not require a self-stabilizing solution, the agent can easily visit all nodes within $2m$ steps with a simple depth first traversal (DFT). Panaite and Pelc [11] gave a faster algorithm, whose cover time is $m + 3n$ steps. They assume that the nodes are labeled by the unique identifiers. Their algorithm uses $O(m \log n)$ bits in the agent-memory, while it does not use whiteboards. Sudo, Baba, Nakamura, Ooshita, Kakugawa, and Masuzawa [15] gave another implementation of this

Table 1 Randomized self-stabilizing graph exploration algorithms for a single agent.

	Expected Cover Time	Agent Memory	Memory on node v
Simple Random Walk	$O(\min(mn, mD \log n))$	0	0
Biased Random Walk [7] (require $\Delta \geq \delta_{\max}$)	$O(n^2 \delta_{\max} \log n)$	$O(\log \Delta)$	$O(\delta_v \log \Delta)$
		$O(\Delta \log \Delta)$	0
$\mathcal{R}_c$ (require $c \geq 2$)	$O\left(m \cdot \min\left(D, \frac{n}{c}+1, \frac{D}{c}+\log n\right)\right)$	$O(\log c)$	$O(\log(\delta_v + c))$

Table 2 Deterministic self-stabilizing graph exploration algorithms for a single agent.

	Cover Time	Agent Memory	Memory on node v
Rotor-router [13]	$O(mD)$	0	$O(\log \delta_v)$
$\texttt{UXS}_N$ [14] (require $N \geq n$)	polynomial in N	$O(\log N)$	0
$\texttt{SUXS}_N$ [17] (require $N \geq n$)	polynomial in n	$O(\log N)$	0
2-color DFT [8]	$O(mD)$	$O(1)$	$O(\log \delta_v)$
$\mathcal{D}_k$ (require $k \geq \max(D, \delta_{\max})$)	$O(m + nD)$	$O(\log k)$	$O(\log k)$

algorithm: they removed the assumption of the unique identifiers and reduced the space complexity on the agent-memory from $O(m \log n)$ bits to $O(n)$ bits by using $O(n)$ bits in each whiteboard. It is worthwhile to mention that these algorithms [11, 15] guarantee the termination of the agent after exploration is completed, whereas designing a self-stabilizing exploration algorithm with termination is impossible. Self-stabilization and termination contradict each other by definition: if an agent-state that yields termination exists, the agent never completes exploration when starting exploration with this state. If such state does not exist, the agent never terminates the exploration.

In the classical or standard distributed computing model (excluding mobile agents), the self-stabilizing token circulation problem, particularly self-stabilizing depth-first token circulation (DFTC), has been extensively studied [6, 3, 12]. Introduced by Huang and Chen [6], this problem was addressed with a self-stabilizing DFTC algorithm using $O(\log n)$ bits per process, which was later reduced to $O(\log \delta_{\max})$ bits by Datta, Johnen, Petit, and Villain[3]. Petit [12] developed a time-optimal (*i.e.,* $O(n)$-time) self-stabilizing DFTC algorithm that also requires $O(\log n)$ bits per process. However, these algorithms are not directly applicable to self-stabilizing exploration by a single agent because the network models are fundamentally different. In the standard model, n computational processes can communicate with each other via communication links in parallel, whereas in our model, only a single agent computes and updates the states of nodes in the network, potentially requiring more time to solve problems. On the other hand, one of the main challenges for self-stabilizing token circulation in the standard model is maintaining exactly one token starting from any configuration where there maybe no tokens or where two or more tokens may exist. As mentioned above, this challenge does not apply to our model, where there is always a single agent in any configuration. Yet, many techniques from standard distributed computing might be adaptable for mobile agent algorithms. For example, our self-stabilizing exploration algorithms employ the technique of repeatedly recoloring graph nodes to resolve variable inconsistencies, a common approach in the design of self-stabilizing algorithms (see Dolev, Israeli, and Moran [5]).

1.2 Our Contribution

In this paper, we investigate how short a cover time we can achieve in a self-stabilizing setting. One can easily observe that the cover time is lower bounded by $\Omega(m)$: any deterministic algorithm requires $\Omega(m)$ steps and any randomized algorithm requires $\Omega(m)$ steps in expectation before the agent visits all nodes. (For completeness, we will prove this lower bound as Theorem 3) Our goal is to give an algorithm whose cover time is close to this lower bound with as small complexity of agent-memory and whiteboards as possible.

We give two self-stabilizing exploration algorithms $\mathcal{R}_c$ and $\mathcal{D}_k$, where c and k are the design parameters. The cover times and the space complexities of the proposed algorithms and the existing algorithms are summarized in Tables 1 and 2.

Algorithm $\mathcal{R}_c$ is a randomized algorithm, where the agent visits all nodes within $O\left(m \cdot \min\left(D, \frac{n}{c}+1, \frac{D}{c}+\log n\right)\right)$ steps in expectation and uses $O(\log c)$ bits in the agent-memory and $O(\log \delta + \log c)$ bits of the whiteboard of each node with degree δ. Thus, we have trade-off between the cover time and the space complexity. The larger c we use, the smaller cover time we obtain. In particular, the expected cover time is $O(m \log n)$ steps if we set $c = \Omega(D/\log n)$, and it becomes optimal (*i.e.,* $O(m)$ steps) if we set $c = \Omega(n)$. This means that we require the knowledge of $\Omega(n)$ value to make $\mathcal{R}_c$ time-optimal. Fortunately, this assumption can be ignored from a practical point of view: even if c is extremely larger than n, the overhead will be just an additive factor of $\log c$ in the space complexity. Thus, any large $c \in \mathit{poly}(n) \cap \Omega(n)$ is enough to obtain the optimal cover time and the space complexity of $O(\log n)$ bits both in the agent memory and whiteboards. Moreover, irrespective of parameter $c \geq 2$, the cover time is $O(mD)$ steps with probability 1.

Algorithm $\mathcal{D}_k$ is a deterministic algorithm. The cover time of $\mathcal{D}_k$ is $O(m + nD)$ steps, which does not depend on parameter k, while the agent uses $O(\log k)$ bits both for the agent-memory and the whiteboard of each node. Thus, we do not have trade-off between the cover time and the space complexity. However, unlike $\mathcal{R}_c$, we require an upper bound on the diameter and the maximum degree of the graph, that is, $\mathcal{D}_k$ requires $k \geq \max(D, \delta_{\max})$ to solve a self-stabilizing exploration. If $k < \max(D, \delta_{\max})$, the correctness of $\mathcal{D}_k$ is no longer guaranteed. However, the knowledge of an upper bound on $\max(D, \delta_{\max})$ is not a strong assumption because the space complexity increases only logarithmically in k: we can assign any large $\mathit{poly}(n)$ value for k to satisfy $k \geq \max(D, \delta_{\max})$ while keeping the space complexity of the agent-memory and v's whiteboard bounded by $O(\log n)$ bits. For example, consider the case that we set $k = 2^{500}$. Then, $\mathcal{D}_k$ can fail only if $D \geq 2^{500}$, which is too large to consider in practice. This extremely large value for k results in the increase of the memory usage only by 500 bits in the agent and whiteboards.

It remains open if there is a deterministic self-stabilizing exploration algorithm with optimal cover time, *i.e.,* $O(m)$ steps.

2 Preliminaries

Let $G = (V, E, p)$ be a simple, undirected, and connected graph where V is the set of nodes and E is the set of edges. The edges are locally labeled at each node: we have a family of functions $p = (p_v)_{v \in V}$ such that each $p_v : \{\{v, u\} \mid u \in N(v)\} \to \{0, 1, \ldots, \delta_v - 1\}$ uniquely assigns a *port number* to every edge incident to node v. Two port numbers $p_u(e)$ and $p_v(e)$ are independent of each other for edge $e = \{u, v\} \in E$.

An *algorithm* is defined as a 3-tuple $\mathcal{P} = (\phi, \mathcal{M}, \mathcal{W})$, where $\mathcal{M}$ is the set of states for the agent, $\mathcal{W} = (\mathcal{W}_k)_{k \in \mathbb{N}}$ is the family such that $\mathcal{W}_k$ is the set of states for each node with degree k, and ϕ is a function that determines how the agent updates its state (*i.e.,* agent memory) and the state of the current node (*i.e., whiteboard*). At each time step, the agent is located at exactly one node $v \in V$, and moves through an edge incident to v. Every node $v \in V$ has a whiteboard $w(v) \in \mathcal{W}_{\delta_v}$, which the agent can access freely when it visits v. The function ϕ is invoked every time the agent visits a node or when the exploration begins. Suppose that the agent with state $s \in \mathcal{M}$ has moved to node v with state $w(v) = x \in \mathcal{W}_{\delta_v}$ from $u \in N(v)$. Let $p_{\mathrm{in}} = p_v(\{u, v\})$. The function ϕ takes 4-tuple $(\delta_v, p_{\mathrm{in}}, s, x)$ as the input and returns 3-tuple $(p_{\mathrm{out}}, s', x')$ as the output. Then, the agent updates its state to s' and $w(v)$ to x', after which it moves via port p_{out}, that is, it moves to v' such that $p_{\mathrm{out}} = p_v(\{v, v'\})$. At the beginning of an execution, we let p_{in} be an arbitrary integer in $\{0, 1, \dots, \delta_v - 1\}$ where v is the node that the agent exists on. Note that if algorithm $\mathcal{P}$ is randomized one, function ϕ returns the probabilistic distributions for $(p_{\mathrm{out}}, s', x')$.

Given a graph $G = (V, E)$, a *configuration* (or a global state) consists of the location of the agent, the state of the agent (including p_{in}), and the states of all the nodes in V. Algorithm $\mathcal{P}$ is a self-stabilizing exploration algorithm for a class $\mathcal{G}$ of graphs if for any graph $G = (V, E, p) \in \mathcal{G}$, the agent running $\mathcal{P}$ on G eventually visits all the nodes in V at least once starting from any configuration. Note that, by the above definition, any self-stabilizing exploration algorithm ensures that the single agent visits every node infinitely often.

We measure the efficiency of algorithm $\mathcal{P}$ by three metrics: *the cover time*, *the agent memory space*, and *the whiteboard memory space*. All the above metrics are evaluated in the worst-case manner with respect to graph G and an initial configuration. The cover time is defined as the number of moves that the agent makes before it visits all nodes. If algorithm $\mathcal{P}$ is a randomized one, the cover time is evaluated in expectation. The memory spaces of the agent and the whiteboard on node v are just defined as $\log_2 |\mathcal{M}|$ and $\log_2 |\mathcal{W}_{\delta_v}|$, respectively.

3 Main Theorems

The main theorems of this paper are listed below. Due to page limitations, we omit the descriptions of algorithms $\mathcal{R}_c$ and $\mathcal{D}_k$, as well as the proofs for these theorems. Please see the arXiv version [16] for the detailed algorithms and proofs.

▶ **Theorem 1.** *Algorithm $\mathcal{R}_c$ is a randomized self-stabilizing exploration algorithm for all simple, undirected, and connected graphs. Irrespective of c, the cover time is $O(mD)$ steps with probability 1. The expected cover time is $O\left(m \cdot \min\left(D, \frac{n}{c} + 1, \frac{D}{c} + \log n\right)\right)$ steps. The agent memory space is $O(\log c)$ and the memory space of each node v is $O(\log c + \log \delta_v)$.*

▶ **Theorem 2.** *Algorithm $\mathcal{D}_k$ is a deterministic self-stabilizing exploration algorithm for all simple, undirected, and connected graphs with a diameter and maximum degree of at most k. The cover time is $O(m + nD)$ steps, regardless of the value of k. The memory requirement is $O(\log k)$ for both the agent and each node.*

▶ **Theorem 3.** *Let $\mathcal{P}$ be any exploration algorithm. For any positive integers n, m such that $n \geq 3$ and $n - 1 \leq m \leq n(n+1)/2$, there exits a simple, undirected, and connected graph $G = (V, E)$ with $|V| = n$ and $|E| = m$ such that the agent running $\mathcal{P}$ on G starting from some node in V requires at least $(m-1)/4$ steps to visit all nodes in V in expectation.*

References

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

2 Lélia Blin, Maria Gradinariu Potop-Butucaru, and Sébastien Tixeuil. On the self-stabilization of mobile robots in graphs. In *International Conference On Principles Of Distributed Systems*, pages 301–314. Springer, 2007. `doi:10.1007/978-3-540-77096-1_22`.

3 Ajoy K Datta, Colette Johnen, Franck Petit, and Vincent Villain. Self-stabilizing depth-first token circulation in arbitrary rooted networks. *Distributed Computing*, 13(4):207–218, 2000. `doi:10.1007/PL00008919`.

4 Edsger W Dijkstra. Self-stabilization in spite of distributed control. In *Selected writings on computing: a personal perspective*, pages 41–46. Springer, 1982.

5 Shlomi Dolev, Amos Israeli, and Shlomo Moran. Uniform dynamic self-stabilizing leader election. *IEEE Transactions on Parallel and Distributed Systems*, 8(4):424–440, 1997. `doi:10.1109/71.588622`.

6 Shing-Tsaan Huang and Nian-Shing Chen. Self-stabilizing depth-first token circulation on networks. *Distributed Computing*, 7(1):61–66, 1993. `doi:10.1007/BF02278857`.

7 Satoshi Ikeda, Izumi Kubo, and Masafumi Yamashita. The hitting and cover times of random walks on finite graphs using local degree information. *Theoretical Computer Science*, 410(1):94–100, 2009. `doi:10.1016/J.TCS.2008.10.020`.

8 Toshimitsu Masuzawa and Sébastien Tixeuil. Quiescence of self-stabilizing gossiping among mobile agents in graphs. *Theoretical computer science*, 411(14-15):1567–1582, 2010. `doi:10.1016/J.TCS.2010.01.006`.

9 Peter Matthews. Covering problems for markov chains. *The Annals of Probability*, 16(3):1215–1228, 1988.

10 Fukuhito Ooshita, Ajoy K Datta, and Toshimitsu Masuzawa. Self-stabilizing rendezvous of synchronous mobile agents in graphs. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 18–32. Springer, 2017. `doi:10.1007/978-3-319-69084-1_2`.

11 P. Panaite and A. Pelc. Exploring unknown undirected graphs. *Journal of Algorithms*, 33(2):281–295, 1999. `doi:10.1006/JAGM.1999.1043`.

12 Franck Petit. Fast self-stabilizing depth-first token circulation. In *International Workshop on Self-Stabilizing Systems*, pages 200–215. Springer, 2001. `doi:10.1007/3-540-45438-1_14`.

13 Vyatcheslav B Priezzhev, Deepak Dhar, Abhishek Dhar, and Supriya Krishnamurthy. Eulerian walkers as a model of self-organized criticality. *Physical Review Letters*, 77(25):5079, 1996.

14 O Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):1–24, 2008. `doi:10.1145/1391289.1391291`.

15 Yuichi Sudo, Daisuke Baba, Junya Nakamura, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. A single agent exploration in unknown undirected graphs with whiteboards. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 98(10):2117–2128, 2015. `doi:10.1587/TRANSFUN.E98.A.2117`.

16 Yuichi Sudo, Fukuhito Ooshita, and Sayaka Kamei. Self-stabilizing graph exploration by a single agent, 2020. `arXiv:2010.08929`.

17 Amnon Ta-Shma and Uri Zwick. Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences. *ACM Transactions on Algorithms (TALG)*, 10(3):1–15, 2014. `doi:10.1145/2601068`.

18 Vladimir Yanovski, Israel A Wagner, and Alfred M Bruckstein. A distributed ant algorithm for efficiently patrolling a network. *Algorithmica*, 37(3):165–186, 2003. `doi:10.1007/S00453-003-1030-9`.