Self-Stabilizing Fully Adaptive Maximal Matching

Shimon Bitton
Intel Corporation, Haifa, Israel

Yuval Emek
Technion – Israel Institute of Technology, Haifa, Israel

Taisuke Izumi
Osaka Unversity, Japan

Shay Kutten
Technion – Israel Institute of Technology, Haifa, Israel

Abstract

A self-stabilizing randomized algorithm for mending *maximal matching (MM)* in synchronous networks is presented. Starting from a legal MM configuration and assuming that the network undergoes k faults or topology changes (that may occur in multiple batches), the algorithm is guaranteed to stabilize back to a legal MM configuration in time $O(\log k)$ in expectation and with high probability (in k), using constant size messages. The algorithm is simple to implement and is uniform in the sense that it does not assume unique identifiers, nor does it assume any global knowledge of the communication graph including its size. It relies on a generic *probabilistic phase synchronization* technique that may be useful for other self-stabilizing problems. The algorithm compares favorably with the existing self-stabilizing MM algorithms in terms of the dependence of its run-time on k, a.k.a. *fully adaptive run-time*. In fact, this dependence is asymptotically optimal for uniform algorithms that use constant size messages.[1]

2012 ACM Subject Classification Computer systems organization → Fault-tolerant network topologies

Keywords and phrases self-stabilization, maximal matching, fully adaptive run-time, dynamic graphs

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2024.33

Related Version *Full Version*: `https://arxiv.org/abs/2105.09756` [18]

Funding *Yuval Emek*: supported by the Bernard M. Gordon Center for Systems Engineering at the Technion.
Taisuke Izumi: supported by JSPS KAKENHI Grant Numbers 23H04385 and 22H03569.
Shay Kutten: supported by ISF grant 1346/22 and by the Grand Technion Energy Program.

1 Introduction

Distributed systems are notoriously subject to faults and much of the effort in the design and implementation of distributed algorithms is devoted to fault tolerance issues. Introduced in the seminal paper of Dijkstra [26], *self-stabilization* is a fundamental approach to fault tolerance requiring that the system recovers from any number of transient faults and stabilizes back to a legal configuration within finite time. The strong guarantees of this natural approach attracted a lot of attention over the years making self-stabilization an extensively studied research field (see [27, 4] for textbooks).

[1] This paper is extracted from (a much longer) full version [18]. The main contribution of [18] is a self-stabilizing black-box transformer that can be applied to LCL problems in general. The self-stabilizing MM algorithm developed in the current paper is closely related to the one obtained by invoking the transformer on the (fault free) algorithm presented in Sec. 4.1, although the run-time and message size bounds of the former are better than those of the latter.

The main performance measure for self-stabilizing algorithms is their stabilization *run-time.* With very few exceptions (discussed in Sec. 1.3), this is measured as a function of the size of the system: the more processors are in the system, the longer is the time it takes for the algorithm to stabilize. However, while self-stabilizing systems are guaranteed to recover from *any* number of transient faults, in most systems, the common scenario is that over a limited time interval, only *few* faults occur. Although a small number of faults can seriously hinder the operation of the whole system [51], one may hope that the system recovers from them faster than the recovery time from a larger number of faults, regardless of the total number of processors.

With this hope in mind, we turn to the notion of *fully adaptive run-time* that expresses the recovery time of a self-stabilizing algorithm as a function of the number of faults (supporting also dynamic *topology changes*), rather than the size of the system. Our goal in the current paper is to investigate this appealing complexity measure in the context of the classic *maximal matching (MM)* problem as we now turn to define.

1.1 Model, Problem, and Complexity Measure

Consider a distributed algorithm `Alg` running on a message passing communication network represented as a simple undirected graph $G = (V, E)$. Under `Alg`, the nodes operate in synchronous *rounds* so that in each round $t \in \mathbb{Z}_{\geq 0}$, every node $v \in V$ executes the following operations: (1) v performs local computation that includes reading from and writing to its registers; (2) v sends messages to (a subset of) its neighbors; and (3) v receives the messages sent to it by its neighbors in round t. Round t is assumed to span the time interval $[t, t+1)$ so that time t marks the beginning of the round. The *configuration* of `Alg` at time t is an abstract object that encodes the content of all nodes' registers at that time.

Let $N(v)$ be the set of neighbors of a node $v \in V$ and let $\mathrm{d}(v) = |N(v)|$ be its degree. We adhere to the conventions of the *port numbering* model of distributed graph algorithms where each node $v \in V$ is associated with an (arbitrarily chosen) *port bijection* $p_v : [\mathrm{d}(v)] \to N(v)$ so that from v's perspective, node $p_v(i)$ is referred to as neighbor i for each $i \in [\mathrm{d}(v)]$. Assuming that $p_v(i) = u$ and $p_u(j) = v$, when v (resp., u) sends a message to its neighbor i (resp., j), this message is written into a designated *port register* $\pi_u(j)$ (resp., $\pi_v(i)$) that u (resp., v) can read during the local computation step of the subsequent round. We do not assume that v (resp., u) knows anything about the identity of its neighbor i (resp., j) otherwise. In fact, in the current paper, we focus on *uniform* algorithms, namely, the nodes do not have unique identifiers, nor do they hold any global knowledge of the graph G (including its size).

Self-Stabilizing Maximal Matching. An edge subset $M \subseteq E$ is a *matching* in G if the degree of every node in the graph (V, M) is at most 1. A matching M is *maximal* if M' is not a matching for any $M \subset M' \subseteq E$. We assume that the interface of each node $v \in V$ in a distributed maximal matching (*MM*) algorithm includes a designated *matching register* $\mu_v \in \{\bot\} \cup \{\mathtt{U}\} \cup \{1, \ldots, \mathrm{d}(v)\}$ whose semantics is as follows: $\mu_v = \bot$ indicates that v is still undecided; $\mu_v = \mathtt{U}$ indicates that v is (decided to be) unmatched; and $\mu_v = i$ indicates that v is matched to its neighbor i.

To support the consistency of the nodes' matching registers, the interface of node v also includes, for each $i \in [\mathrm{d}(v)]$, a designated *(matching) status field* $\pi_v^\sigma(i) \in \{\bot\} \cup \{\mathtt{U}, \mathtt{H}, \mathtt{E}\}$, within the port register $\pi_v(i)$, whose role is to "reflect" the content of $\mu_{p_v(i)}$. Specifically, assuming that $p_v(i) = u$ and $p_u(j) = v$, the semantics of the status field $\pi_v^\sigma(i)$ is as follows: $\pi_v^\sigma(i) = \bot$ indicates that $\mu_u = \bot$; $\pi_v^\sigma(i) = \mathtt{U}$ indicates that $\mu_u = \mathtt{U}$; $\pi_v^\sigma(i) = \mathtt{H}$ indicates that $\mu_u = j$ (`H` stands for '(matched) here'); and $\pi_v^\sigma(i) = \mathtt{E}$ indicates that $\mu_u \in [\mathrm{d}(u)] - \{j\}$ (`E` stands for '(matched) elsewhere').

Assuming that $p_v(i) = u$ and $p_u(j) = v$, nodes v and u are said to be *strongly matched* if (1) $\mu_v = i$ and $\mu_u = j$; and (2) $\pi_v^\sigma(i) = \pi_u^\sigma(j) = \mathtt{H}$. Node v is said to be *strongly unmatched* if (1) $\mu_v = \mathtt{U}$; (2) $\pi_v^\sigma(i) = \mathtt{E}$ for all $i \in [\mathrm{d}(v)]$; and (3) every node $u \in N(v)$ is strongly matched. We say that a configuration of a distributed MM algorithm `Alg` is *legal* if every node $v \in V$ is either strongly matched or strongly unmatched, observing that the strongly matched nodes induce a maximal matching in G; we refer to this maximal matching as the *output* of `Alg`.

A distributed algorithm is *self-stabilizing* if it is guaranteed to reach a legal configuration in finite time from any initial configuration (see [26, 27]). This notion is captured by a malicious *adversary*, who knows the algorithm but is oblivious to its coin tosses, that determines the algorithm's initial configuration.

For reasons that will become clear soon, in the current paper, the adversary is allowed to "intervene" in the algorithm's execution also after time 0 (corresponding to the initial configuration) and modify the content of any register maintained by the algorithm including the aforementioned port registers $\pi_v(\cdot)$ (and their status fields $\pi_v^\sigma(\cdot)$) and matching registers μ_v. The adversary can also impose dynamic *topology changes* including the addition (resp., removal) of nodes and edges which are reflected in adding new port registers $\pi_v(\cdot)$ (resp., removing existing port registers) and updating the port bijections p_v. In this regard, a node $v \in V$ is said to be *manipulated* (by the adversary) in round t if at least one of the following three events occur during round t: (i) the content of (any of) v's registers is modified; (ii) the port bijection p_v is modified; or (iii) v is added to the graph as a new node. (Notice that condition (ii) includes topological changes involving any edges incident on v as well as "rewiring" of v's ports.)

Fully Adaptive Run-Time. Let `Alg` be a distributed self-stabilizing MM algorithm. Consider times $0 \leq t^*_{\mathrm{adv}} < t^*_0$ and integer $k > 0$ and suppose that (1) `Alg` is in a legal configuration at time t^*_{adv}; (2) the adversary manipulates k nodes during the time interval $[t^*_{\mathrm{adv}}, t^*_0)$; and (3) the adversary does not manipulate any node from time t^*_0 onwards. We say that `Alg` has *fully adaptive run-time* $T(k)$ for some function $T : \mathbb{Z}_{>0} \to \mathbb{Z}_{>0}$ if it is guaranteed that there exist some time t and maximal matching M such that (i) `Alg` is in a legal configuration with output M from time t onwards; and (ii) $t \leq t^*_0 + T(k)$. If t (and M) are random variables determined by the coin tosses of `Alg`, then we require that condition (ii) holds in expectation and with high probability, where throughout this paper, the term with high probability (abbreviated *w.h.p.*) refers to events that occur with probability at least $1 - k^{-c}$ for a constant c that can be made arbitrarily large.

It is important to point out that the number k of manipulated nodes is chosen by the adversary and is not known to the algorithm that also does not know any bound on k.

1.2 Our Contribution

Our main contribution is cast in the following theorem.

▶ **Theorem 1.1.** *There exists a randomized uniform self-stabilizing MM algorithm that uses messages of size $O(1)$ whose fully adaptive run-time is $O(\log k)$.*

The reader may have noticed that the guarantees of our algorithm do not depend in any way on the size of the graph G. Indeed, Theorem 1.1 holds also for *infinite graphs* as long as the node degrees are finite (though not necessarily bounded).[2] We believe that distributed

[2] In infinite graphs, the notion of fully adaptive run-time can be extended to support an infinite number of manipulations as long as the manipulated nodes can be partitioned into clusters of size at most k

computation in infinite graphs is a fascinating research domain and advocate using the notion of (fully) adaptive run-time as a natural complexity measure when it comes to self-stabilizing algorithms in this domain.

As established in [48], any uniform (non-self-stabilizing) distributed MM algorithm with constant size messages requires $\Omega(\log n)$ time in expectation when running on rings (and paths) of size n. This implies that any uniform self-stabilizing MM algorithm with constant size messages requires $\Omega(\log k)$ time in expectation to stabilize from $k \leq n$ transient faults, which means that the fully adaptive run-time bound promised in Thm. 1.1 is asymptotically optimal.

1.3 Related Work and Discussion

Self-stabilizing symmetry breaking is an extensively studied research topic, see, e.g., [38] for a survey. In particular, there is a long line of works on self-stabilizing algorithms for maximal matching (and closely related problems) [42, 64, 21, 22, 40, 37, 35, 36, 57, 58, 63, 7, 46, 24, 5, 43, 65, 15]. Under synchronous schedules (assumed in the current paper), the state-of-the-art MM self-stabilization run-time upper bounds are $O(\log n)$, obtained by the (randomized uniform) algorithm of Turau [65], and $O(\Delta + \log^* n)$, obtained by the (deterministic non-uniform) algorithm of Barenboim et al. [15], where n is the number of nodes in the graph and Δ is its maximum degree. Notice that those run-time upper bounds are not fully adaptive.

The MM problem received a lot of attention also in the literature on fault free (non-self-stabilizing) distributed graph algorithms, see, e.g., [45, 44, 54, 39, 61, 16, 32, 30, 12, 33]. Some of the algorithmic ideas in the current paper are inspired by ideas originated in that literature, e.g., [44]. Another source of inspiration is the algorithm of [66] for the related task of approximating maximum matching.

The self-stabilizing research community has a special interest in generic transformers that translate non-self-stabilizing algorithms into self-stabilizing ones, see, e.g., [10, 8, 9, 2, 1, 53, 13]. None of the existing transformers though can produce self-stabilizing algorithms that use constant size messages and run in sub-diameter time (as the algorithm developed in the current paper does). Moreover, with the exception of the transformer presented in the full version [18], none of the existing transformers (provably) guarantees a fully adaptive stabilization run-time, regardless of the graph's size and diameter or any probabilistic assumption on the distribution of faults.

There has been a lot of interest in saving run-time in dynamic models by adapting to the actual load on the algorithm. In the non-self-stabilizing computing context, Lamport suggested that an algorithm should benefit from the common case where the number of contending processes is small [52]. A maximum matching approximation can be mended following a single topological change in constant distributed time [55]. A distributed algorithm that mends a maximal independent set after a single change in constant time was presented in [20]. Mending other local functions in a similar setting, also in constant distributed time, was addressed in [47]. Mending after a small number of changes was addressed in [59]. In the context of distributed fault tolerance, a similar motivation stands behind the notions of obstruction freedom [41].

In highly dynamic graphs, mending of various functions has been treated recently [14, 19]. Comparing these papers to the current one, their algorithms are not self-stabilizing. Moreover, the authors of [19] optimize the *amortized* time complexity (vs. the worst case time complexity

which are sufficiently far away from each other.

used in the current paper) assuming that the graph is initially empty, whereas the run-time of the algorithm of [14] is expressed as a (logarithmic) function of n rather than k (so it is not adaptive). Both papers rely on messages of super-constant size.

For non-distributed algorithms, the area of dynamic algorithms started with addressing algorithms that already computed a complete and correct output (say a minimum spanning tree [31]) and need to mend the output following a single change to the input. A non-distributed update of MM in dynamic graphs for the deletion or the removal of a single edge is performed in amortized (expected) $O(\log n)$ time [17] or worst case (deterministic) $O(\sqrt{m})$ time, where m denotes the number of edges [60].

In the context of self stabilization, the notion of super-stabilization [28] is concerned with decreasing the impact of topological changes on the complexity of self-stabilizing algorithms and the notion of containment [34] has been used to ensure that if only a single fault occurred, then the complexity is smaller than in the general fault case. For the maximal independent set problem, multiple simultaneous changes to the input were handled in [50] assuming a model that is only partially self-stabilizing (the adversary may change the data structure, but not other registers, e.g., the program counter). Moreover, while the run-time measure there is adaptive, expressed as a function of the number k of faults, it is not *fully* adaptive since all the faults are assumed to occur in a single batch before the mending starts. In contrast, the run-time analysis in the current paper is fully adaptive in the sense that it allows the adversary to divide the k faults (and/or topology changes) over multiple batches.

Algorithms whose time adaptivity was at least linear in the number of faults for various tasks were suggested in [25, 6, 49, 11].

1.4 Phase Based Distributed Algorithms

A common design pattern in distributed algorithms working under the synchronous fault free model is to divide the execution into *phases* so that each phase is composed of a fixed number ϕ of single round *steps*, grouped together to fulfill a task that cannot be achieved in a single round. The execution then progresses by running the phases in succession, where step $0 \leq j \leq \phi - 1$ of phase $i = 0, 1, \dots$ is executed in round $t = i\phi + j$. A key property of this design pattern is that the nodes execute their phases in synchrony so that step j of phase i is executed by all nodes in the same round.

More often than not, the algorithm runs the same code in each phase which means that the nodes do not have to keep track of the current round t; rather, it suffices that they have access to a *step oracle*, denoted hereafter by `Get_Step`, that simply returns the current step $j = t \bmod \phi$. We refer to the family of such algorithms as *phase based* algorithms and note that this family includes the distributed graph algorithm "classics" of [23] (3-coloring of trees), [56, 3] (maximal independent set), and [44] (maximal matching), as well as more recent distributed graph algorithms, see, e.g., [16, 29, 62, 32].

The step oracle discussion is usually avoided in the literature on synchronous fault free algorithms as these algorithms can easily maintain a register that keeps track of $t \bmod \phi$. This is not the case in the self-stabilizing setting though: the adversary may modify this register, causing the nodes to execute their phases "out of sync" indefinitely. Indeed, we cannot assume that the nodes in a self-stabilizing algorithm have free access to a step oracle, hence the adaptation of a phase based algorithm designed to operate in a fault free environment to the self-stabilization setting imposes a significant challenge (cf. [65]).

To overcome this challenge, we introduce a technique called *probabilistic phase synchronization*. In high level terms (refer to Sec. 3 for a formal exposition), this technique replaces the step oracle by a generic (algorithm independent) module, referred to as `PPS`, whose implementation is based on a simple ergodic Markov chain, maintained independently at each node. The state space of the Markov chain is $\{\hbar, -1, 0, 1, \ldots, \phi - 1\}$, where states $0, 1, \ldots, \phi - 1$ are identified with the ϕ steps of the phase and $\hbar$ and -1 are auxiliary states whose role is explained in the sequel.

Using the `PPS` module, the adaptation of a fault free phase based algorithm `Alg` into a self-stabilizing algorithm becomes almost a "black-box compiler" based on the following two modifications: (i) Replace each call to `Get_Step` by a call to a procedure named `Get_Step_PPS` that returns the current state of `PPS`. (ii) When executing the phase, node v restricts its communication to its *phase synchronized* neighbors, namely, adjacent nodes whose `PPS` state agrees with that of v.

We design the `PPS` module so that (a) if nodes u and v start a phase in the same round, then they remain phase synchronized until the phase ends; and (b) the Markov chain admits a poly(ϕ) mixing time, ensuring that every pair of adjacent nodes become phase synchronized sufficiently often. As we show in the sequel (see [18] for more details), these two properties guarantee progress on behalf of the algorithm, allowing us to bound the stabilization time. While our analysis is specific to maximal matching, the aforementioned adaptation is fairly generic and serves as the cornerstone for the transformer discussed in the full version [18].

2 Preliminaries

The algorithms are presented from the perspective of a node $v \in V$ and for clarity of the exposition, they are described as if v can address its neighbors directly, rather than through the proxy of the port bijection $p_v(\cdot)$. Specifically, we use $In_v(u)$ to denote the message received by v from neighbor $u \in N(v)$, recalling that this message is actually written into the port register $\pi_v(p_v^{-1}(u))$ and that v does not know u beyond its (local) port number $p_v^{-1}(u)$. Likewise, we define m_v to be the neighbor pointed to by the matching register μ_v if $\mu_v \in [\mathrm{d}(v)]$; and $m_v = \mu_v$ otherwise (i.e., if $\mu_v = \perp$ or $\mu_v = \mathtt{U}$). Implementing our algorithm based on the proxy of the port bijections (as defined in Sec. 1.1) is straightforward.

For a node $v \in V$ and a register $\mathtt{reg}_v$ of v, let $\mathtt{reg}_{v,t}$ denote the value of $\mathtt{reg}_v$ at time t, that is, at the beginning of round t prior to any local computation. Notice that if v is not manipulated during the time interval $[t-1, t]$, as is guaranteed for every $t > t_0^*$, then the value $\mathtt{reg}_{v,t}$ is also the value of $\mathtt{reg}_v$ at the end of round $t - 1$.

Given some graph H and a node v in H, we use the notation $N_H(v)$ and $\mathrm{d}_H(v)$ for v's neighbor set and degree, respectively, in H.

3 Probabilistic Phase Synchronization

Let `Alg` be a phase based (fault free) distributed algorithm with phase length ϕ. We assume that under `Alg`, each node $v \in V$ calls `Get_Step` in every round and proceeds according to the step number $j \in \{0, 1, \ldots, \phi - 1\}$ it receives. Our goal in this section is to apply the probabilistic phase synchronization "compiler" to `Alg`, obtaining an algorithm `Alg'` that has no access to `Get_Step`.

Consider a node $v \in V$. The design of `Alg'` (from `Alg`) is based on the `PPS` module that can be viewed as an ergodic Markov chain over the state space $\{\hbar, -1, 0, 1, \ldots, \phi - 1\}$ that v runs, independently of the other nodes. The current state of (v's copy of) `PPS` is stored in a register denoted by $step_v$; to minimize the interface between `Alg'` and the generic module `PPS`, the access of `Alg'` to $step_v$ is made by means of procedure `Get_Step_PPS` that simply returns the current value of this register. The main idea is that each call of `Alg` to `Get_Step` is replaced by a call to `Get_Step_PPS` under `Alg'`.

Algorithm 1 `Step_Counter_`ϕ(), code for node v. Executed in every round at the end of the local computation stage.

```
1: if $step_v = \hbar$ then $step_v \leftarrow -1$ w.p. $1/2$ ; $\hbar$ w.p. $1/2$
2: else if $step_v = \phi - 1$ then $step_v \leftarrow \hbar$
3: else $step_v \leftarrow step_v + 1$
```

Beside `Get_Step_PPS`, the `PPS` module has one additional procedure whose role is to implement the Markov chain, advancing it in every round. This procedure is denoted by `Step_Counter_`ϕ (see Algorithm 1) and it is assumed to be invoked by `PPS` in every round, towards the end of the local computation stage, after any call to `Get_Step_PPS` has already been made. We emphasize that `Step_Counter_`ϕ is the only procedure that writes to the $step_v$ register; any other access of `Alg′` to this register is restricted to the read-only procedure `Get_Step_PPS`. The fundamental properties of `Step_Counter_`ϕ are summarized in the following observation (see Fig. 1 for an illustration of the underlying Markov chain).

▶ **Observation 3.1.** *For every time $t \geq t_0^*$ and node $v \in V$, we have*
- $\mathbb{P}(step_{v,t+1} = j \mid step_{v,t} = j - 1) = 1$ *for every* $0 \leq j \leq \phi - 1$*;*
- $\mathbb{P}(step_{v,t+1} = \hbar \mid step_{v,t} = \phi - 1) = 1$*; and*
- $\mathbb{P}(step_{v,t+1} = \hbar \mid step_{v,t} = \hbar) = \mathbb{P}(step_{v,t+1} = -1 \mid step_{v,t} = \hbar) = 1/2$.

This holds independently of any coin toss of v prior to time t and of any coin toss of all other nodes.

Consider some round $t \in \mathbb{Z}_{\geq 0}$ and for $j \in \{-1, 0, 1, \ldots, \phi - 1\}$, let

$$\Sigma_t^j = \{u \in V \mid step_{u,t} = j\} .$$

We say that adjacent nodes $u, v \in V$ are *phase synchronized* in round t if they both belong to Σ_t^j for some $j \in \{-1, 0, 1, \ldots, \phi - 1\}$. Notice that node v cannot be phase synchronized (with any node) in round t if $step_{v,t} = \hbar$.

Fix some node $v \in V$. If $v \in \Sigma_t^j$ for some $j \in \{0, 1, \ldots, \phi - 1\}$, then v runs in round t under `Alg′` the same code that `Alg` runs in step j with one crucial difference: v progresses as if it operates in the graph induced by Σ_t^j. In other words, for the sake of the simulation of `Alg`, node v restricts its interaction to its phase synchronized neighbors, that is, the nodes in $N(v) \cap \Sigma_t^j$, ignoring any node in $N(v) - \Sigma_t^j$. Notice that v still sends messages to *all* its neighbors, however the messages sent to the nodes in $N(v) - \Sigma_t^j$ do not carry any `Alg`-related information.

To make this possible, each node $u \in V$ appends the value of $step_{u,t-1}$ to its outgoing messages (sent to all its neighbors) in round $t - 1$, thus allowing its neighbors $v \in N(u)$ to determine if they are phase synchronized with u in round t. Indeed, Obs. 3.1 ensures that if the message that a node v receives from its neighbor u in round t indicates that $step_{u,t-1} = j - 1$ for some $0 \leq j \leq \phi - 1$, then v can deduce that $step_{u,t} = j$ unless the adversary manipulates u or v during the time interval $[t - 1, t]$.

The role of state -1 is to allow the nodes to identify their phase synchronized neighbors already in step $j = 0$ so that they can simulate a full phase of `Alg`. Specifically, if $u \in \Sigma_t^{-1}$, then node u does not do anything in round t other than appending $step_{u,t} = -1$ to its outgoing messages, allowing any neighbor $v \in \Sigma_t^{-1}$ to identify that it is phase synchronized with u in the next round $t + 1$.

Similarly to state -1, state $\hbar$ is also special in the sense that it does not correspond to any step in `Alg` and as such, corresponds to an "empty code": If $step_{u,t} = \hbar$, then node u does not do anything in round t other then appending $step_{u,t} = \hbar$ to its outgoing messages.

Algorithm 2 `Finalize_and_Send_Messages()`, code for node v.

```
1: if $m_v = \bot$ or $m_v = \texttt{U}$ then
2:     for all $u \in N(v)$ do $Out_v^\sigma(u) \leftarrow m_v$
3: else
4:     $Out_v^\sigma(m_v) \leftarrow \texttt{H}$
5:     for all $u \in N(v) - \{m_v\}$ do $Out_v^\sigma(u) \leftarrow \texttt{E}$
6: for all $u \in N(v)$ do Send $Out_v(u)$ to $u$
```

We now turn to bound the mixing time of the Markov chain associated with PPS. To this end, let $S_\phi = \{\hbar, -1, 0, 1, \ldots, \phi - 1\}$ denote its state space. The following lemma is established by showing that this ergodic Markov chain belongs to a family of Markov chains that has been identified and analyzed in [67]; its proof is deferred to Appendix A.

▶ **Lemma 3.2.** *Fix some time $t_0 \geq t_0^*$, node v, and $j_0 \in S_\phi$. For every $0 < \epsilon < 1$, there exists a time $\hat{t} = t_0 + O(\log(1/\epsilon) \cdot \phi^3)$ such that for every $t \geq \hat{t}$, it holds that*

$$\mathbb{P}\left(step_{v,t} = j \mid step_{v,t_0} = j_0\right) \geq \begin{cases} \frac{2}{\phi+3} \cdot (1-\epsilon)\,, & \text{if } j = \hbar \\ \frac{1}{\phi+3} \cdot (1-\epsilon)\,, & \text{otherwise} \end{cases} .$$

This holds independently of any coin toss of v prior to time t_0 and of any coin toss of all other nodes.

▶ **Corollary 3.3.** *Fix some time $t_0 \geq t_0^*$, node v, and $j_0 \in S_\phi$. For $\epsilon = 1 - \frac{\phi+3}{2\phi}$ and $\tau = O(\phi^3)$, it holds that $\mathbb{P}\left(step_{v,t_0+\tau} = -1 \mid step_{v,t_0} = j_0\right) \geq \frac{1}{2\phi}$. This holds independently of any coin toss of v prior to time t_0 and of any coin toss of all other nodes.*

4 A Self-Stabilizing MM Algorithm

In this section, we develop the self-stabilizing MM algorithm promised in Thm. 1.1, referred to as SSMM. First, we present a phase based fault free MM algorithm, referred to as FFMM, similar in its spirit to the classic distributed algorithm of Israeli and Itai [44].[3] Next, we apply our probabilistic phase synchronization technique to FFMM and slightly modify it to obtain SSMM.

We use $Out_v(u)$ to denote the message sent from v to u; the algorithms are described so that the content of this register is constructed (during the local computation) gradually until it is sent to u. Recall that every message includes a status field, referred to hereafter as the σ-field, that is part of the node's interface (see Sec. 1.1). The status field reflects the node's matching register and it is set before the message is sent (see Algorithm 2). The messages include additional fields on top of the σ-field; we use a superscript x to refer to the x-field of the message so that $Out_v^x(u)$ and $In_v^x(u)$ denote the x-fields in the messages sent from v to u and received by v from u, respectively. For ease of reference, Table 1 summarizes the various registers of FFMM and SSMM and the values they can hold.

[3] Israeli and Itai [44] present their MM algorithm under the CRCW-PRAM model, but by now, it is better known in its implementation as a distributed message passing algorithm (cf. [66]).

Algorithm 3 `FFMM`(), code for node v.

```
for all $u \in N(v)$ do $Out_v^r(u) = \bot$
if $m_v = \bot$ and $In_v^\sigma(u) = \mathtt{E}$ for all $u \in N(v)$ then $m_v \leftarrow \mathtt{U}$
if $m_v = \bot$ then
    if Get_Step() $= 0$ then
        $chosen_v \leftarrow \bot$
        $active_v \leftarrow true$ w.p. $1/2$ ; $false$ w.p. $1/2$
        if $active_v$ then
            $P_v \leftarrow \{u \in N(v) \mid In_v^\sigma(u) = \bot\}$
            if $|P_v| > 0$ then
                $chosen_v \leftarrow$ node picked from $P_v$ uniformly at random
                $Out_v^r(chosen_v) \leftarrow \mathtt{Mreq}$
    if Get_Step() $= 1$ then
        if $active_v = false$ then
            $P_v \leftarrow \{u \in N(v) \mid In_v^{\sigma,r}(u) = (\bot, \mathtt{Mreq})\}$
            if $|P_v| > 0$ then
                $chosen_v \leftarrow$ node picked arbitrarily from $P_v$
                $Out_v^r(chosen_v) \leftarrow \mathtt{Acc}$
    if Get_Step() $= 2$ then
        if $active_v$ then
            if $chosen_v \neq \bot$ and $In_v^r(chosen_v) = \mathtt{Acc}$ then $m_v \leftarrow chosen_v$
        else if $chosen_v \neq \bot$ then $m_v \leftarrow chosen_v$
Finalize_and_Send_Messages()
```

4.1 Algorithm `FFMM`

Like the algorithm of [44], algorithm `FFMM`, presented in Algorithm 3, is phase based with a phase of length 3. In round 0 of the phase, each undecided node v tosses an unbiased coin that determines if v is active or passive (for the duration of the current phase); active nodes then send a matching request to an undecided neighbor picked uniformly at random. In round 1 of the phase, a passive node v that received a matching request from at least one neighbor picks one such neighbor u (arbitrarily) and replies that u's matching request is accepted. Finally, in round 2 of the phase, each edge over which a matching request was accepted is added to the output MM. To implement this logic, the messages sent by `FFMM` use one additional field (on top of the aforementioned σ-field), namely, the r-field that takes values in $\{\mathtt{Mreq}, \mathtt{Acc}\}$ (stands for 'matching request' and 'accept', respectively).

4.2 Algorithm `SSMM`

Our self-stabilizing algorithm `SSMM`, presented in Algorithm 5, is obtained from `FFMM` through the following two modifications: (1) We add an *error detection* subroutine, presented in Algorithm 4, which is invoked in every round (see line 2 in Algorithm 5). In this subroutine, node v checks whether its available local information indicates that it is either strongly matched or strongly unmatched; if v is neither, then it becomes undecided by setting $m_v \leftarrow \bot$. (2) We apply the probabilistic phase synchronization "compiler" of Sec. 3. To this end, the outgoing messages are augmented with an additional s-field (stands for 'step'). This field is used to communicate v's current step so that $Out_v^s(\cdot)$ is set to the value of $step_v$ before the message is sent by calling `Get_Step_PPS` (see line 23 in Algorithm 5).

Algorithm 4 `Detect()`, code for node v.

```
1: if $m_v \in N(v)$ and $In_v^{\sigma}(m_v) = \mathtt{H}$ then return
2: if $m_v = \mathtt{U}$ and $In_v^{\sigma}(u) = \mathtt{E}$ for all $u \in N(v)$ then return
3: $m_v \leftarrow \perp$
```

Algorithm 5 `SSMM()`, code for node v.

```
1: for all $u \in N(v)$ do $Out_v^r(u) = \perp$
2: Detect()
3: if $m_v = \perp$ and $In_v^{\sigma}(u) = \mathtt{E}$ for all $u \in N(v)$ then $m_v \leftarrow \mathtt{U}$
4: if $m_v = \perp$ then
5:     if Get_Step_PPS() $= 0$ then
6:         $chosen_v \leftarrow \perp$
7:         $active_v \leftarrow true$ w.p. $1/2$ ; $false$ w.p. $1/2$
8:         if $active_v$ then
9:             $P_v \leftarrow \{u \in N(v) \mid In_v^{\sigma,s}(u) = (\perp, -1)\}$
10:            if $|P_v| > 0$ then
11:                $chosen_v \leftarrow$ node picked from $P_v$ uniformly at random
12:                $Out_v^r(chosen_v) \leftarrow \mathtt{Mreq}$
13:    if Get_Step_PPS() $= 1$ then
14:        if $active_v = false$ then
15:            $P_v \leftarrow \{u \in N(v) \mid In_v^{\sigma,r,s}(u) = (\perp, \mathtt{Mreq}, 0)\}$
16:            if $|P_v| > 0$ then
17:                $chosen_v \leftarrow$ node picked arbitrarily from $P_v$
18:                $Out_v^r(chosen_v) \leftarrow \mathtt{Acc}$
19:    if Get_Step_PPS() $= 2$ then
20:        if $active_v$ then
21:            if $chosen_v \neq \perp$ and $In_v^r(chosen_v) = \mathtt{Acc}$ then $m_v \leftarrow chosen_v$
22:        else if $chosen_v \neq \perp$ then $m_v \leftarrow chosen_v$
23: for all $u \in N(v)$ do $Out_v^s(u) \leftarrow$ Get_Step_PPS()
24: Finalize_and_Send_Messages()
```

5 Analysis

Assume that SSMM is in a legal configuration at time t^*_{adv} and that the adversary manipulates $k > 0$ nodes during the time interval $[t^*_{\text{adv}}, t^*_0)$ and does not manipulate any node from time t^*_0 onwards. Our goal in this section is to establish Thm. 1.1 by proving that SSMM stabilizes to a legal configuration by time $t^*_0 + O(\log k)$ in expectation and w.h.p. (in k).

Recall that the phase length of SSMM is $\varphi = \phi + 1$, where $\phi = 3$ is the phase length of FFMM. Although the analysis presented in this section is dedicated to SSMM, it is performed with respect to a general parameter ϕ (and φ) so that it can be applied to self-stabilizing algorithms derived from other phase based algorithms. Since the adversary may add/remove nodes/edges, the graph may change during the time interval $[t^*_{\text{adv}}, t^*_0)$; in what follows, we reserve $G = (V, E)$ for the graph that exists at time t^*_0, recalling that this is also the graph at any time $t > t^*_0$.

5.1 Outline

This section presents the general structure of SSMM's analysis. It hinges on Prop. 5.1–5.6 whose combination yields Thm. 1.1. We start with Prop. 5.1 ensuring that once the algorithm reaches a legal configuration, it stays in a legal configuration.

▶ **Proposition 5.1.** *For every $t \geq t_0^*$ and $v \in V$, if v is strongly matched (resp., unmatched) at time t, then v is strongly matched (resp., unmatched) at time $t + 1$.*

We say that v is *locally matched* to node $u \in N(v)$ if (i) $m_v = u$; and (ii) $In_v^\sigma(u) = \mathtt{H}$; when the identity of u is not important, we may refer to v as being *locally matched* without explicitly mentioning u. Notice that by definition, if v is strongly matched, then v is also locally matched. Given some $t \geq t_0^*$, let V_t be the set of nodes which are *not* locally matched at time t, let $E_t = E \cap (V_t \times V_t)$, and let $G_t = (V_t, E_t)$ be the graph induced on G by V_t. When combined with Prop. 5.1, the following proposition implies that we do not have to worry about the nodes that are no longer in G_t.

▶ **Proposition 5.2.** *For every $t > t_0^*$ and $v \in V$, if v is locally matched at time t, then v is strongly matched at time t.*

By combining Prop. 5.1 and 5.2 with the following proposition, we deduce that if $E_t = \emptyset$, then SSMM is in a legal configuration at time $t + 1$ and will remain so indefinitely.

▶ **Proposition 5.3.** *For every $t > t_0^*$ and $v \in V_t$, if $\mathrm{d}_{G_t}(v) = 0$, then v is strongly unmatched at time $t + 1$.*

The rest of the analysis is devoted to showing that it takes $O(\log k)$ rounds in expectation and w.h.p. for the set E_t to become empty. Let $\tau = O(\phi^3)$ be the parameter promised in Cor. 3.3. The next proposition implies that from time $t > t_0^*$, algorithm SSMM stabilizes in $O\left(\frac{\phi^5}{\log(1+1/\phi^2)} \log |E_t| + \log k\right)$ rounds in expectation and w.h.p. (in k), which is $O(\log |E_t| + \log k)$ when ϕ is a constant as in SSMM. This means in particular that once SSMM reaches a configuration in which $|E_t| \leq \mathrm{poly}(k)$, it takes $O(\log k)$ additional rounds in expectation and w.h.p. for the algorithm to stabilize.

▶ **Proposition 5.4.** *Fix some time $t > t_0^*$ and a configuration at time t. There exist universal positive constants α and δ such that $\mathbb{P}\left(|E_{t+\tau+\varphi}| \leq (1 - \frac{\delta}{\phi^2}) \cdot |E_t|\right) \geq \alpha/\phi^2$.*

Unfortunately, by manipulating k nodes, the adversary can lead the algorithm to a graph $G_{t_0^*}$ whose edge set is arbitrarily large (and not polynomially bounded) with respect to k. To resolve this obstacle, we prove that it takes the algorithm $O(\log k)$ rounds in expectation and w.h.p. to reduce the number of edges in G_t down to $O(k^2)$. This relies on the following proposition derived from a nice combinatorial property of maximal matching.

▶ **Proposition 5.5.** *Let $S = \{v \in V \mid v$ is strongly matched at time $t_0^*\}$. There exists a node set $K \subseteq V - S$ of size $|K| \leq 2k$ such that $I = V - (S \cup K)$ is an independent set in G.*

Consider the sets K and I promised in Prop. 5.5 and some positive constant $\xi = \xi(\phi)$ whose value is determined later on. For $t > t_0^*$, let $D_t = \{v \in K \cap V_t \mid \mathrm{d}_{G_t}(v) \geq \xi k\}$ and let T be the random variable that takes on the earliest time t such that $D_t = \emptyset$. Since I is an independent set, every edge in E_T is incident on at least one vertex in K, thus $|E_T| < |K| \cdot \xi k \leq 2\xi k^2$. Recalling that ϕ is a constant in SSMM, the proof of Thm. 1.1 is completed due to the following proposition, implying, by standard probabilistic arguments (see Appendix B), that $T \leq O(\log k)$ in expectation and w.h.p.

▶ **Proposition 5.6.** *Fix some time $t > t_0^*$ and a configuration at time t. There exists a universal positive constant δ such that $\mathbb{P}(v \notin D_{t+2\varphi+1}) \geq \delta 2^{-\phi}$ for every node $v \in D_t$.*

5.2 Correctness of the Detection Process

We begin the journey towards proving Prop. 5.1–5.6 by establishing two simple structural observations regarding the operation of SSMM.

▶ **Observation 5.7.** *For every time $t > t_0^*$, node $v \in V$, and node $u \in N(v)$, the following three statements are logically equivalent: (1) $m_{v,t} = u$; (2) $In^{\sigma}_{u,t}(v) = \mathtt{H}$; (3) $In^{\sigma}_{w,t}(v) = \mathtt{E}$ for every $w \in N(v) - \{u\}$.*

▶ **Observation 5.8.** *For every $t > t_0^*$ and $v \in V$, if v is strongly matched at time t, then v is strongly matched at time $t + 1$.*

Proof. Since v is strongly matched at time t, it holds that $m_{v,t} = u$, $m_{u,t} = v$, and $In^{\sigma}_{u,t}(v) = In^{\sigma}_{v,t}(u) = \mathtt{H}$. The design of SSMM ensures that in this case, $m_v = u$ and $m_u = v$ after the call to Detect in line 2, thus $m_v = u$ and $m_u = v$ before the call to Finalize_and_Send_Messages in line 24 and $m_{v,t+1} = u$ and $m_{u,t+1} = v$. By Obs. 5.7, it holds that u and v are strongly matched at time $t + 1$. ◀

We are now ready to prove Prop. 5.1–5.3.

Proof of Prop. 5.1. The statement regarding a strongly matched node is proved by Obs. 5.8. If v is a strongly unmatched at time t, then $m_{v,t} = \mathtt{U}$, $In^{\sigma}_{v,t}(u) = \mathtt{E}$ for every $u \in N(v)$, and all neighbors of v are strongly matched. The design of SSMM ensures that in this case, $m_v = \mathtt{U}$ after the call to Detect in line 2, hence $m_v = \mathtt{U}$ at the end of the round and $m_{v,t+1} = \mathtt{U}$. By Obs. 5.8, all of v's neighbors remain strongly matched, and by Obs. 5.7 we know that, $In^{\sigma}_{v,t+1}(u) = \mathtt{E}$ for every $u \in N(v)$. ◀

Proof of Prop. 5.2. Since v is locally matched to u at time t, it holds that $m_{v,t} = u$ and $In^{\sigma}_{v,t}(u) = \mathtt{H}$. By Obs. 5.7, we know that $m_{u,t} = v$ and $In^{\sigma}_{u,t}(v) = \mathtt{H}$. ◀

Proof of Prop. 5.3. Since $\mathrm{d}_{G_t}(v) = 0$ we know that all of v's neighbors are locally matched. By Prop. 5.2, all of v's neighbors are also strongly matched. Since v is not locally matched we conclude that none of v's neighbors are strongly matched to him. In this case, by Obs. 5.7, $In^{\sigma}_{v,t}(u) = \mathtt{E}$, for every $u \in N(v)$. Moreover, by Prop. 5.1 and Obs. 5.7 it holds that, for every $u \in N(v)$, u is strongly matched at time $t + 1$ and $In^{\sigma}_{v,t+1}(u) = \mathtt{E}$.

If $m_{v,t} \in N(v)$, then after the call to Detect in Line 2 of SSMM it holds that $m_v = \perp$. Thus, in Line 3 of SSMM node v sets $m_v = \mathtt{U}$. This value will not change during round t, thus $m_{v,t+1} = \mathtt{U}$.

If $m_{v,t} = \mathtt{U}$, then $m_v = \mathtt{U}$ after the call to Detect in Line 2 of SSMM. The design of SSMM ensures that in that case $m_v = \mathtt{U}$ in the end of round t, thus $m_{v,t+1} = \mathtt{U}$. Otherwise ($m_{v,t} = \perp$), in Line 3 of SSMM node v sets $m_v = \mathtt{U}$. This value will not change during round t, thus $m_{v,t+1} = \mathtt{U}$. We can conclude that v is strongly unmatched at time $t + 1$. ◀

5.3 Eliminating the Graph

Our goal in this section is to establish Prop. 5.4– 5.6, starting with some additional definitions. For every time $t \geq t_0^*$, let

$$\widetilde{V}_t = V_t \cap \{v \in V \mid step_{v,t} = -1\}$$

be the subset of nodes that start a phase in round t, recalling that $step_{v,t}$ is the state of v's PPS module at time t. Let $\widetilde{E}_t = E_t \cap (\widetilde{V}_t \times \widetilde{V}_t)$ and let $\widetilde{G}_t = (\widetilde{V}_t, \widetilde{E}_t)$ be the graph induced on G_t by $\widetilde{V}_t$.

A key feature of `SSMM` is that in each phase, node $v \in V$ writes to $\texttt{reg}_v$ before reading from it for every register $\texttt{reg}_v$ other than m_v and $In_v(\cdot)$. Therefore, we can conceptually assume that these registers are reset to an arbitrary default value at step -1.

Consider some graph $H = (V_H, E_H)$. Node $v \in V_H$ is said to be *good* in H if at least $1/3$ of its neighbors u in H have degree $\mathrm{d}_H(u) \leq \mathrm{d}_H(v)$. The following lemma is established by Alon et al. [3, Lem. 4.4].

▶ **Lemma 5.9.** *Let H be an undirected graph. At least $1/2$ of the edges in H are incident on a good node.*

Lem. 5.9 is exploited in the following two lemmas to show that sufficiently many edges are removed with a sufficiently high probability; Prop. 5.4 follows by a standard Markov inequality argument (see Appendix B).

▶ **Lemma 5.10.** *Fix some time $t > t_0^*$ and a configuration at time t. There exists a universal positive constant p_g such that $\mathbb{P}\left(v \notin V_{t+\varphi}\right) \geq p_g$ for every good node $v \in \widetilde{V}_t$.*

Proof. Denote by d the degree of node v in $\widetilde{G}_t$, i.e., $d = \mathrm{d}_{\widetilde{G}_t}(v) > 0$. Node v is good, thus there exist $u_1, \ldots, u_{\lceil d/3 \rceil} \in N_{\widetilde{G}_t}(v)$ such that $d_i = \mathrm{d}_{\widetilde{G}_t}(u_i) \leq d$ for every $1 \leq i \leq \lceil d/3 \rceil$. Recall that by the definition of $\widetilde{G}_t$, nodes v and $u_1, \ldots, u_{\lceil d/3 \rceil}$ start a phase (in synchrony) at time t. Node v marks itself as passive in step 0 of the phase with probability $1/2$; condition hereafter on this event. For $1 \leq i \leq \lceil d/3 \rceil$, let B_i be the event that u_i marks itself as active and sends a `Mreq` message to v in step 1 of the phase, noticing that $\mathbb{P}(B_i) = \frac{1}{2d_i} \geq \frac{1}{2d}$. Since the events B_i are independent, it follows that the probability that none of them occurs is up-bounded by $(1 - 1/(2d))^{\lceil d/3 \rceil} < e^{-1/6}$. The assertion follows since the occurrence of any of the events B_i implies that v becomes locally matched by the end of the phase that lasts for φ rounds. ◀

▶ **Lemma 5.11.** *Fix some time $t > t_0^*$ and a configuration at time t. There exists a universal positive constant α such that $\mathbb{E}\left(|E_{t+\tau+\varphi}|\right) \leq (1 - \frac{\alpha}{\phi^2}) \cdot |E_t|$.*

Proof. Denote $m = |E_t|$. Let $\hat{V} \subseteq V_t$ be the (random) set of nodes $v \in V_t$ such that $step_{v,t+\tau} = -1$ and let $\hat{E} = \{\{u, v\} \mid v, u \in \hat{V}, u \neq v\}$. For every edge $e \in E_t$ let A_e be the event that $e \in \hat{E}$. By Cor. 3.3, for every edge $e = (u, v) \in E_t$, we have

$$\mathbb{P}\left(A_e\right) \geq \frac{1}{4\phi^2}. \tag{1}$$

Hence,

$$\mathbb{E}\left(|\hat{E}|\right) \geq \frac{1}{4\phi^2} m. \tag{2}$$

Let $E_{t+\tau}^c = E_t - E_{t+\tau}$ and notice that $E_{t+\tau}^c \cap E_{t+\tau+\varphi} = \emptyset$. We partition the set $\hat{E}$ into two (possibly empty) disjoint sets, $\hat{E} \cap E_{t+\tau}$ and $\hat{E}^c = \hat{E} \cap E_{t+\tau}^c$. By the definition of $\widetilde{E}_{t+\tau}$, it holds that $\hat{E} \cap E_{t+\tau} = \widetilde{E}_{t+\tau}$.

By Lem. 5.9, in the (random) graph $\widetilde{G}_{t+\tau}$ at least half of the edges are incident on a good node (at least one). Every good node in $\widetilde{G}_{t+\tau}$ is removed with probability at least p_g, where p_g is the constant promised in Lem. 5.10. Hence, the expected number of removed edges in the time interval $[t + \tau, t + \tau + \varphi)$ can be low-bounded as follows

$$\mathbb{E}\left(|E_{t+\tau} - E_{t+\tau+\varphi}|\right) = \mathbb{E}\left(\mathbb{E}\left(|E_{t+\tau} - E_{t+\tau+\varphi}| \mid |\widetilde{E}_{t+\tau}|\right)\right) \geq \frac{1}{2} \cdot p_g \cdot \mathbb{E}\left(|\widetilde{E}_{t+\tau}|\right).$$

By definition, $|\hat{E}| = |\hat{E}^c| + |\widetilde{E}_{t+\tau}|$, hence $\mathbb{E}\left(|\hat{E}|\right) = \mathbb{E}\left(|\hat{E}^c|\right) + \mathbb{E}\left(|\widetilde{E}_{t+\tau}|\right)$ and by Eq. 2, $\mathbb{E}\left(|\hat{E}^c|\right) + \mathbb{E}\left(|\widetilde{E}_{t+\tau}|\right) \geq \frac{1}{4\phi^2}m$. This allows us to low-bound the expected number of edges removed in the time interval $[t, t+\tau+\varphi)$ by

$$\begin{aligned}\mathbb{E}\left(|E_t - E_{t+\tau+\varphi}|\right) &\geq \mathbb{E}\left(|\hat{E}^c|\right) + \mathbb{E}\left(|E_{t+\tau} - E_{t+\tau+\varphi}|\right) \geq \mathbb{E}\left(|\hat{E}^c|\right) + \frac{1}{2} \cdot p_g \cdot \mathbb{E}\left(|\widetilde{E}_{t+\tau}|\right) \\ &\geq \frac{1}{2} \cdot p_g \cdot \left(\mathbb{E}\left(|\hat{E}^c|\right) + \mathbb{E}\left(|\widetilde{E}_{t+\tau}|\right)\right) \geq \frac{1}{8\phi^2} \cdot p_g \cdot m\,,\end{aligned}$$

thus completing the proof. ◀

We now turn to establishing Prop. 5.5.

Proof of Prop. 5.5. Let A be the set of nodes in $V - S$ that are manipulated (by the adversary) during the time interval $[t^*_{\text{adv}}, t^*_0)$. Let R be the set of nodes in $V - (S \cup A)$ that are strongly matched at time t^*_{adv}, referred to hereafter as *orphans*. We define $K = A \cup R$ and establish the assertion by proving that (1) $|K| \leq 2k$; and (2) $I = V - (S \cup K)$ is an independent set in G.

For every orphan $v \in R$, let $w(v)$ be the node with which v is strongly matched at time t^*_{adv}. Note that $w(v)$ does not necessarily exist in G as it may have been removed during the time interval $[t^*_{\text{adv}}, t^*_0)$. Since v and $w(v)$ are no longer strongly matched at time t^*_0 and since v is not manipulated during the time interval $[t^*_{\text{adv}}, t^*_0)$, Obs. 5.8 implies that $w(v)$ must be manipulated during that time interval. We conclude that $|R| \leq k$ as the mapping defined by w is injective. The bound $|K| \leq 2k$ follows since $|A| \leq k$.

It remains to show that I is an independent set in G. This is done by arguing that every node $v \in I$ is strongly unmatched at time t^*_{adv}. This establishes the assertion recalling that by definition, the nodes in I are not manipulated during the time interval $[t^*_{\text{adv}}, t^*_0)$, hence the adversary does not introduce new edges in $I \times I$. To that end, assume by contradiction that there exists some node $v \in I$ that is not strongly unmatched at time t^*_{adv}. Since `SSMM` is in a legal configuration at time t^*_{adv}, it follows that v must be strongly matched at that time. But by definition, the nodes in $V - S$ that are strongly matched at time t^*_{adv} belong to either R or A, in contradiction to $v \in I = V - (S \cup A \cup R)$. ◀

▶ **Corollary 5.12.** *Fix some time $t > t^*_0$ and a configuration at time t. If $v \in \widetilde{V}_t \cap K$ and $\mathrm{d}_{\widetilde{G}_t}(v) \geq 3k$, then v is good in $\widetilde{G}_t$.*

Proof. Let $d = \mathrm{d}_{\widetilde{G}_t}(v)$. Node v has at most $|K| - 1 < 2k$ neighbors from the set K, thus at least $d - 2k \geq (1/3)d$ of v's neighbors in $\widetilde{G}_t$ belong to $\widetilde{V}_t - K = I \cap \widetilde{V}_t$. By Prop. 5.5, every node $u \in I \cap \widetilde{V}_t$ has degree at most $2k$ which completes the proof. ◀

The analysis is completed by establishing Prop. 5.6.

Proof of Prop. 5.6. For every node $v \in V_t$, let $t \leq t(v)$ be the first time after time t such that $step_{v,t(v)} = \hbar$. According to the definition of the `PPS` module it must hold that $t \leq t(v) \leq t+\varphi$ and notice that $t(v)$ is fully determined by $step_{v,t}$. Denote by A_v the event that $step_{v,t'} = \hbar$ for every $t(v) \leq t' \leq t + \varphi$. By Obs. 3.1, it holds that the event A_v is independent of any coin toss of v prior to time $t(v)$ and of any coin toss of all other nodes and that

$$\mathbb{P}\left(A_v\right) \geq 2^{-\varphi}. \tag{3}$$

For every node $v \in V_t$, we augment the power of the adversary by allowing it choose the outcome of any coin toss in the time interval $[t, t(v))$. Notice that this adversary can only choose the outcome of coin tosses that are within a phase and cannot choose the outcome of coin tosses of the PPS module. Let S be the set of nodes $v \in V_t$ that are not locally matched at time $t(v)$, i.e., $S = \{v \in V_t \mid v \in V_{t(v)}\}$. Let $G^S = (S, E^S)$ be the graph induced on G_t by S.

Fix some node $v \in D_t$. If node v is locally matched at time $t(v)$ or $\mathrm{d}_{G^S}(v) < \xi k$, then $v \notin D_{t+2\varphi+1}$ with probability 1. Otherwise (v is not locally matched at time $t(v)$ and more than ξk of its neighbors are in S), we will show that with probability $\Omega(2^{-\varphi})$ node v is locally matched at time $t + 2\varphi + 1$ which implies that $v \notin D_{t+2\varphi+1}$.

Let B be the event that $\mathrm{d}_{\widetilde{G}_{t+\varphi+1}}(v) \geq 3k$. We start by showing that there exists $\alpha = \alpha(\varphi)$ such that $\mathbb{P}(B) \geq \alpha$. For every $u \in S$, let $\widetilde{A}_u$ be the event that A_u occurred and $step_{v,t+\varphi+1} = -1$. By Eq. 3 and Obs. 3.1 we conclude that $\mathbb{P}\left(\widetilde{A}_u\right) = \mathbb{P}(A_u) \cdot (1/2) \geq 2^{-(\varphi+1)}$. Moreover, the events in $\{\widetilde{A}_u \mid u \in S\}$ are independent.

Denote $d = \mathrm{d}_{G^S}(v)$ and let $u_1, \ldots, u_d$ be the neighbors of v is G^S. Let Y be the random variable that counts the number of occurrences of events $\widetilde{A}_{u_i}$. Notice that if $Y \geq 3k$ and $\widetilde{A}_v$ occurs, then event B occur; moreover, events $Y \geq 3k$ and $\widetilde{A}_v$ are independent and $\mathbb{E}(Y) \geq \xi k 2^{-(\varphi+1)}$ since $d \geq \xi k$.

By choosing $\xi = 4/2^{-(\varphi+1)} = 2^{\varphi+3}$ and applying Chernoff's (lower tail) bound we conclude that

$$\mathbb{P}(Y < 3k) = \mathbb{P}\left(Y < (1 - 1/4) \cdot \xi k 2^{-(\varphi+1)}\right) \leq \mathbb{P}(Y < (1 - 1/4) \cdot \mathbb{E}(Y)) < e^{-k/8} \leq e^{-1/8}.$$

Thus $\mathbb{P}(B) \geq (1 - e^{-1/8}) \cdot 2^{-(\varphi+1)}$. By Cor. 5.12 and Lem. 5.10, occurrence of B implies that v is good in $\widetilde{G}_{t+\phi+1}$ and a good node is removed with at least a constant probability by the end of the phase, i.e., at time $t + 2\varphi + 1$. We conclude that

$$\begin{aligned}\mathbb{P}(v \notin D_{t+2\varphi+1}) &= \mathbb{P}(B \wedge v \notin V_{t+2\varphi+1}) \\ &\geq \mathbb{P}\left(v \notin V_{t+2\varphi+1} \,\middle|\, B\right) \cdot \mathbb{P}(B) = p_g \cdot (1 - e^{-1/8}) \cdot 2^{-(\varphi+1)},\end{aligned}$$

thus establishing the assertion. ◀

▶ Remark. Unfortunately, we did not manage to prove Prop. 5.6 based on the promise of Cor. 3.3 as we did in the proof of Lem. 5.11. Instead, we used the weaker promise of Obs. 3.1 that results in an exponential, rather than polynomial, dependency on the phase length ϕ. As ϕ is a constant in SSMM, this does not affect its asymptotic run-time.

References

1 Yehuda Afek and Shlomi Dolev. Local stabilizer. *Journal of Parallel and Distributed Computing*, 62(5):745–765, 2002. doi:10.1006/jpdc.2001.1823.

2 Yehuda Afek, Shay Kutten, and Moti Yung. The local detection paradigm and its applications to self-stabilization. *Theoretical Computer Science*, 186(1-2):199–229, 1997. doi:10.1016/S0304-3975(96)00286-1.

3 Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of algorithms*, 7(4):567–583, 1986. doi:10.1016/0196-6774(86)90019-2.

4 Karine Altisen, Stéphane Devismes, Swan Dubois, and Franck Petit. Introduction to distributed self-stabilizing algorithms. *Synthesis Lectures on Distributed Computing Theory*, 8(1):1–165, 2019. doi:10.2200/S00908ED1V01Y201903DCT015.

5 Ozkan Arapoglu and Orhan Dagdeviren. An asynchronous self-stabilizing maximal independent set algorithm in wireless sensor networks using two-hop information. In *2019 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–5. IEEE, 2019. doi:10.1109/ISNCC.2019.8909189.

6 Anish Arora and Hongwei Zhang. LSRP: Local stabilization in shortest path routing. *IEEE/ACM Transactions on Networking*, 14(3):520–531, 2006. doi:10.1145/1143396.1143402.

7 Yuma Asada, Fukuhito Ooshita, and Michiko Inoue. An efficient silent self-stabilizing 1-maximal matching algorithm in anonymous networks. *Journal of Graph Algorithms and Applications*, 20(1):59–78, 2016. doi:10.7155/jgaa.00384.

8 Baruch Awerbuch, Boaz Patt-Shamir, and George Varghese. Self-stabilization by local checking and correction (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 268–277. IEEE, IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185378.

9 Baruch Awerbuch, Boaz Patt-Shamir, George Varghese, and Shlomi Dolev. Self-stabilization by local checking and global reset (extended abstract). In *Distributed Algorithms, 8th International Workshop, WDAG '94, Terschelling, The Netherlands, September 29 - October 1, 1994, Proceedings*, volume 857, pages 326–339. Springer, Springer, 1994. doi:10.1007/BFb0020443.

10 Baruch Awerbuch and George Varghese. Distributed program checking: A paradigm for building self-stabilizing distributed protocols (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 258–267. IEEE, IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185377.

11 Yossi Azar, Shay Kutten, and Boaz Patt-Shamir. Distributed error confinement. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 33–42, 2003. doi:10.1145/872035.872041.

12 Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. Lower bounds for maximal matchings and maximal independent sets. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 481–497. IEEE, 2019. doi:10.1109/FOCS.2019.00037.

13 Alkida Balliu, Juho Hirvonen, Darya Melnyk, Dennis Olivetti, Joel Rybicki, and Jukka Suomela. Local mending. In Merav Parter, editor, *Structural Information and Communication Complexity - 29th International Colloquium, SIROCCO 2022, Paderborn, Germany, June 27-29, 2022, Proceedings*, pages 1–20, 2022. doi:10.1007/978-3-031-09993-9_1.

14 Philipp Bamberger, Fabian Kuhn, and Yannic Maus. Local distributed algorithms in highly dynamic networks. In *2019 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2019, Rio de Janeiro, Brazil, May 20-24, 2019*, pages 33–42, 2019. doi:10.1109/IPDPS.2019.00015.

15 Leonid Barenboim, Michael Elkin, and Uri Goldenberg. Locally-iterative distributed $(\Delta + 1)$-coloring and applications. *J. ACM*, 69(1):5:1–5:26, 2022. doi:10.1145/3486625.

16 Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. *J. ACM*, 63(3):20:1–20:45, 2016. doi:10.1145/2903137.

17 Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in $O(\log n)$ update time. *SIAM Journal on Computing*, 44(1):88–113, 2015. doi:10.1137/130914140.

18 Shimon Bitton, Yuval Emek, Taisuke Izumi, and Shay Kutten. Fully adaptive self-stabilizing transformer for LCL problems. *CoRR*, abs/2105.09756, 2024. doi:10.48550/arXiv.2105.09756.

19 Keren Censor-Hillel, Neta Dafni, Victor I. Kolobov, Ami Paz, and Gregory Schwartzman. Fast and simple deterministic algorithms for highly-dynamic networks. *CoRR*, abs/1901.04008, 2019. doi:10.48550/arXiv.1901.04008.

20 Keren Censor-Hillel, Elad Haramaty, and Zohar Karnin. Optimal dynamic distributed mis. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 217–226. ACM, 2016. doi:10.1145/2933057.2933083.

21 Johanne Cohen, Jonas Lefèvre, Khaled Maâmra, Laurence Pilard, and Devan Sohier. A self-stabilizing algorithm for maximal matching in anonymous networks. *Parallel Process. Lett.*, 26(4):1650016:1–1650016:17, 2016. doi:10.1142/S012962641650016X.

22 Johanne Cohen, George Manoussakis, Laurence Pilard, and Devan Sohier. A self-stabilizing algorithm for maximal matching in link-register model. In *International Colloquium on Structural Information and Communication Complexity*, pages 14–19. Springer, 2018. `doi:10.1007/978-3-030-01325-7_2`.

23 Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986. `doi:10.1016/S0019-9958(86)80023-7`.

24 Ajoy Kumar Datta, Lawrence L. Larmore, and Toshimitsu Masuzawa. Maximum matching for anonymous trees with constant space per process. In *19th International Conference on Principles of Distributed Systems, OPODIS 2015, December 14-17, 2015, Rennes, France*, volume 46 of *LIPIcs*, pages 16:1–16:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. `doi:10.4230/LIPIcs.OPODIS.2015.16`.

25 Murat Demirbas, Anish Arora, Tina Nolte, and Nancy Lynch. A hierarchy-based fault-local stabilizing algorithm for tracking in sensor networks. In *International Conference on Principles of Distributed Systems*, pages 299–315. Springer, 2004. `doi:10.1007/11516798_22`.

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

27 Shlomi Dolev. *Self-stabilization.* MIT press, 2000. URL: `http://www.cs.bgu.ac.il/%7Edolev/book/book.html`.

28 Shlomi Dolev and Ted Herman. Superstabilizing protocols for dynamic distributed systems (abstract). In *Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing, Ottawa, Ontario, Canada, August 20-23, 1995*, page 255. ACM, ACM, 1995. `doi:10.1145/224964.224993`.

29 Michael Elkin and Jian Zhang. Efficient algorithms for constructing (1+epsilon, beta)-spanners in the distributed and streaming models. *Distributed Comput.*, 18(5):375–385, 2006. `doi:10.1007/s00446-005-0147-2`.

30 Manuela Fischer. Improved deterministic distributed matching via rounding. In *31st International Symposium on Distributed Computing, DISC*, pages 17:1–17:15, 2017. `doi:10.4230/LIPIcs.DISC.2017.17`.

31 G.N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. on Computing (SICOMP)*, 14(4):781–798, 1985. `doi:10.1137/0214055`.

32 Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 270–277. SIAM, 2016. `doi:10.1137/1.9781611974331.ch20`.

33 Mohsen Ghaffari. Distributed maximal independent set using small messages. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 805–820. SIAM, 2019. `doi:10.1137/1.9781611975482.50`.

34 Sukumar Ghosh, Arobinda Gupta, Ted Herman, and Sriram V Pemmaraju. Fault-containing self-stabilizing algorithms. In *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, pages 45–54. ACM, 1996. `doi:10.1145/248052.248057`.

35 Wayne Goddard, Stephen T. Hedetniemi, David Pokrass Jacobs, and Pradip K. Srimani. A robust distributed generalized matching protocol that stabilizes in linear time. In *23rd International Conference on Distributed Computing Systems Workshops (ICDCS 2003 Workshops), 19-22 May 2003, Providence, RI, USA*, pages 461–465. IEEE, IEEE Computer Society, 2003. `doi:10.1109/ICDCSW.2003.1203595`.

36 Wayne Goddard, Stephen T. Hedetniemi, David Pokrass Jacobs, and Pradip K. Srimani. Self-stabilizing protocols for maximal matching and maximal independent sets for ad hoc networks. In *17th International Parallel and Distributed Processing Symposium (IPDPS 2003), 22-26 April 2003, Nice, France, CD-ROM/Abstracts Proceedings*, page 162. IEEE, IEEE Computer Society, 2003. `doi:10.1109/IPDPS.2003.1213302`.

37 Maria Gradinariu and Colette Johnen. Self-stabilizing neighborhood unique naming under unfair scheduler. In *European Conference on Parallel Processing*, pages 458–465. Springer, 2001. doi:10.1007/3-540-44681-8_67.

38 Nabil Guellati and Hamamache Kheddouci. A survey on self-stabilizing algorithms for independence, domination, coloring, and matching in graphs. *Journal of Parallel and Distributed Computing*, 70(4):406–415, 2010. doi:10.1016/j.jpdc.2009.11.006.

39 Michal Hanckowiak, Michal Karonski, and Alessandro Panconesi. On the distributed complexity of computing maximal matchings. *SIAM Journal on Discrete Mathematics*, 15(1):41–57, 2001. doi:10.1137/S0895480100373121.

40 Stephen T Hedetniemi, David P Jacobs, and Pradip K Srimani. Maximal matching stabilizes in time o (m). *Information Processing Letters*, 80(5):221–223, 2001. doi:10.1016/S0020-0190(01)00171-5.

41 Maurice Herlihy, Victor Luchangco, and Mark Moir. Obstruction-free synchronization: Double-ended queues as an example. In *23rd International Conference on Distributed Computing Systems (ICDCS 2003), 19-22 May 2003, Providence, RI, USA*, pages 522–529. IEEE, IEEE Computer Society, 2003. doi:10.1109/ICDCS.2003.1203503.

42 Su-Chu Hsu and Shing-Tsaan Huang. A self-stabilizing algorithm for maximal matching. *Information processing letters*, 43(2):77–81, 1992. doi:10.1016/0020-0190(92)90015-N.

43 Can Umut Ileri and Orhan Dagdeviren. A self-stabilizing algorithm for b-matching. *Theoretical Computer Science*, 753:64–75, 2019. doi:10.1016/j.tcs.2018.06.042.

44 Amos Israeli and Alon Itai. A fast and simple randomized parallel algorithm for maximal matching. *Information Processing Letters*, 22(2):77–80, 1986. doi:10.1016/0020-0190(86)90144-4.

45 Richard M Karp and Avi Wigderson. A fast parallel algorithm for the maximal independent set problem. *Journal of the ACM (JACM)*, 32(4):762–773, 1985. doi:10.1145/4221.4226.

46 Masahiro Kimoto, Tatsuhiro Tsuchiya, and Tohru Kikuno. The time complexity of Hsu and Huang's self-stabilizing maximal matching algorithm. *IEICE transactions on information and systems*, 93(10):2850–2853, 2010. doi:10.1587/transinf.E93.D.2850.

47 Michael König and Roger Wattenhofer. On local fixing. In *International Conference On Principles Of Distributed Systems*, pages 191–205. Springer, 2013. doi:10.1007/978-3-319-03850-6_14.

48 Kishore Kothapalli, Christian Scheideler, Melih Onus, and Christian Schindelhauer. Distributed coloring in $O(\log n)$ bit rounds. In *20th International Parallel and Distributed Processing Symposium (IPDPS 2006), Proceedings, 25-29 April 2006, Rhodes Island, Greece*. IEEE, 2006. doi:10.1109/IPDPS.2006.1639281.

49 Shay Kutten and Boaz Patt-Shamir. Time-adaptive self stabilization. In *Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing*, pages 149–158. ACM, 1997. doi:10.1145/259380.259435.

50 Shay Kutten and David Peleg. Fault-local distributed mending (extended abstract). In *Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing, Ottawa, Ontario, Canada, August 20-23, 1995*, pages 20–27. ACM, ACM, 1995. doi:10.1145/224964.224967.

51 Leslie Lamport. Distribution, May 1987. Email message sent to a DEC SRC bulletin board at 12:23:29 PDT on 28 May 87. URL: https://www.microsoft.com/en-us/research/publication/distribution/.

52 Leslie Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems (TOCS)*, 5(1):1–11, 1987. doi:10.1145/7351.7352.

53 Christoph Lenzen, Jukka Suomela, and Roger Wattenhofer. Local algorithms: Self-stabilization on speed. In *Symposium on Self-Stabilizing Systems*, pages 17–34. Springer, 2009. doi:10.1007/978-3-642-05118-0_2.

54 Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992. doi:10.1137/0221015.

55 Zvi Lotker, Boaz Patt-Shamir, and Adi Rosén. Distributed approximate matching. *SIAM Journal on Computing*, 39(2):445–460, 2009. doi:10.1137/080714403.

56 Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM journal on computing*, 15(4):1036–1053, 1986. doi:10.1137/0215074.

57 Fredrik Manne, Morten Mjelde, Laurence Pilard, and Sébastien Tixeuil. A new self-stabilizing maximal matching algorithm. *Theoretical Computer Science*, 410(14):1336–1345, 2009. doi:10.1016/j.tcs.2008.12.022.

58 Fredrik Manne, Morten Mjelde, Laurence Pilard, and Sébastien Tixeuil. A self-stabilizing 23-approximation algorithm for the maximum matching problem. *Theoretical Computer Science*, 412(40):5515–5526, 2011. doi:10.1016/j.tcs.2011.05.019.

59 Darya Melnyk, Jukka Suomela, and Neven Villani. Mending partial solutions with few changes. *CoRR*, abs/2209.05363, 2022. doi:10.48550/arXiv.2209.05363.

60 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. *ACM Trans. Algorithms*, 12(1):7:1–7:15, 2016. doi:10.1145/2700206.

61 Alessandro Panconesi and Romeo Rizzi. Some simple distributed algorithms for sparse networks. *Distributed Comput.*, 14(2):97–100, 2001. doi:10.1007/PL00008932.

62 Alex Scott, Peter Jeavons, and Lei Xu. Feedback from nature: an optimal distributed algorithm for maximal independent set selection. In *Proceedings of the 2013 ACM symposium on Principles of distributed computing*, pages 147–156, 2013. doi:10.1145/2484239.2484247.

63 Zhengnan Shi, Wayne Goddard, and Stephen T Hedetniemi. An anonymous self-stabilizing algorithm for 1-maximal independent set in trees. *Information Processing Letters*, 91(2):77–83, 2004. doi:10.1016/j.ipl.2004.03.010.

64 Gerard Tel. Maximal matching stabilizes in quadratic time. *Information Processing Letters*, 49(6):271–272, 1994. doi:10.1016/0020-0190(94)90098-1.

65 Volker Turau. Making randomized algorithms self-stabilizing. In *Structural Information and Communication Complexity - 26th International Colloquium, SIROCCO 2019, L'Aquila, Italy, July 1-4, 2019, Proceedings*, pages 309–324, 2019. doi:10.1007/978-3-030-24922-9_21.

66 Mirjam Wattenhofer and Roger Wattenhofer. Distributed weighted matching. In *International Symposium on Distributed Computing*, pages 335–348. Springer, 2004. doi:10.1007/978-3-540-30186-8_24.

67 Elizabeth Lee Wilmer. *Exact rates of convergence for some simple non-reversible Markov chains*. PhD thesis, Department of Mathematics, Harvard University, 1999.

A Proving Lemma 3.2

Proof of Lem. 3.2. To avoid cumbersome notation, we assume throughout this proof that $t_0 = 0$. Let $\{X_t\}_{t=0}^{\infty}$ be a stochastic process such that $X_t = \mathit{step}_{v,t}$. The stochastic process $\{X_t\}_{t=0}^{\infty}$ can be defined by the discrete time Markov chain M_ϕ with state space S_ϕ as in Fig. 1.

Let $P_\phi \in \mathbb{R}_{\geq 0}^{S_\phi \times S_\phi}$ be the time-homogeneous transition matrix of the Markov chain M_ϕ and P_ϕ^t the matrix P_ϕ to the power of t. Denote entry $(i,j) \in S_\phi \times S_\phi$ of P_ϕ by $P_\phi(i,j)$ and recall that for every $t, s \in \mathbb{Z}_{\geq 0}$ such that $t \geq 1$ it holds that $P_\phi^t(i,j) = \mathbb{P}(X_{t+s} = j \mid X_s = i)$. For convenience sake, let $n = |S_\phi| = \phi + 2$.

The chain M_ϕ is ergodic, thus there exists a unique stationary distribution on the state space S_ϕ, which we denote by the size n vector $\boldsymbol{\pi} \in \mathbb{R}_{>0}^{S_\phi}$. It is easily verifiable that $\boldsymbol{\pi}(\hbar) = \frac{2}{n+1}$ and $\boldsymbol{\pi}(j) = \frac{1}{n+1}$ for every $j \in S_\phi - \{\hbar\}$. We will prove that there exists $\hat{t} = O(\log(1/\epsilon) \cdot n^3)$ such that for every $t \geq \hat{t}$ it holds that

$$P_\phi^t(j_0, \hbar) \geq \frac{2}{n+1}(1 - \epsilon) \tag{4}$$

and

$$P^t_\phi(j_0, j) \geq \frac{1}{n+1}(1-\epsilon), \forall j \in S_\phi - \{\hbar\}. \tag{5}$$

The chain M_ϕ is a special case of the *coin-flip* chain as defined in [67] with $p = q = 1/2$. By Proposition 5.8, Theorem 3.6 and Lemma 2.4 in [67], for every positive $\epsilon < 1$ there exists a time $t(n) = O(n^3)$ such that for every $t \geq O(\log(1/\epsilon) \cdot t(n))$, and $i \in S_\phi$

$$\max_{j \in S_\phi} \left| \frac{P^t_\phi(i, j)}{\boldsymbol{\pi}(j)} - 1 \right| \leq \epsilon.$$

Thus, for every $t \geq O(\log(1/\epsilon) \cdot n^3)$ we can low-bound Eq. 4–5 by,

$$P^t_\phi(j_0, j) \geq \boldsymbol{\pi}(j) \cdot (1-\epsilon).$$

The proof is completed by the value of $\boldsymbol{\pi}$. ◀

B Standard Probabilistic Arguments

In the section we will show how to prove Thm. 1.1 using Prop. 5.4–5.6. It is also used to prove Prop. 5.4 using Lem. 5.11.

Fix some $m, \ell \in \mathbb{Z}_{>0}$ and $c > 1$. We define $1 < \beta(c) < \frac{1}{1-1/c}$. Let $\{X_i\}_{i=0}^{\infty}$ be a stochastic process such that (1) $X_0 = m$; (2) for every $i \in \mathbb{Z}_{\geq 0}$, it holds that $X_i \geq 0$ with probability 1; and (3) for every $\alpha \in \mathbb{Z}_{>0}$ it holds that $\mathbb{E}\left(X_{\alpha\ell} \,\middle|\, X_{(\alpha-1)\ell} = d\right) \leq (1 - \frac{1}{c}) \cdot d$. The random variable that corresponds to the run time of an algorithm with the aforementioned properties is defined as $T = \min\{i \mid X_i < 1\}$.

Fix some $\alpha \in \mathbb{Z}_{>0}$ and $d \in \mathbb{R}_{\geq 0}$. Let $Y_d \sim X_{\alpha\ell} \mid X_{(\alpha-1)\ell} = d$. By applying Markov inequality we get

$$\mathbb{P}\left(Y_d > \beta(c)(1 - \frac{1}{c})d\right) \leq \mathbb{P}\left(Y_d > \beta(c)\mathbb{E}\left(Y_d\right)\right) < \frac{1}{\beta(c)}.$$

Hence,

$$\mathbb{P}\left(Y_d \leq \beta(c)(1 - \frac{1}{c})d\right) \geq \left(1 - \frac{1}{\beta(c)}\right)$$

This is regardless of the value of d and α. So, for every α with probability at least $\left(1 - \frac{1}{\beta(c)}\right)$ the value of $X_{\alpha\ell}$ decreases by a factor of $\beta(c)(1 - \frac{1}{c}) < 1$ and we denote this event by A_α.

Let $j = \frac{\log m}{\log\left(\frac{1}{\beta(c)(1-1/c)}\right)}$ and $\hat{\alpha}$ denote the jth occurrence of events from the set $\{A_\alpha \mid \alpha \in \mathbb{Z}_{>0}\}$. Since $X_0 = m$, it holds that $X_{\hat{\alpha}\ell} < 1$ implying that $T/\ell \leq \hat{\alpha}$. It is easily verifiable that T/ℓ is stochastically dominated by a random variable $Y \sim NB\left(j, 1 - 1/\beta(c)\right)$ and the proof is completed by the properties of Y.

C Figures and Tables

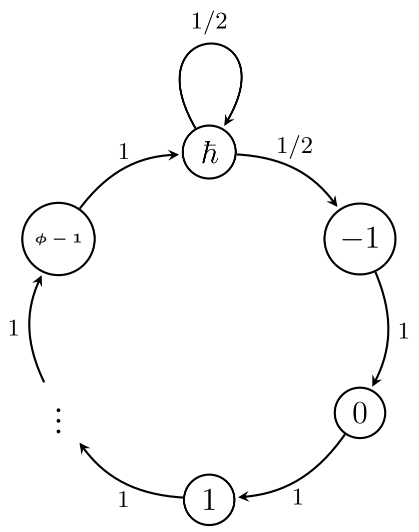

Figure 1 The underlying Markov chain of PPS with state space $S_\phi = \{\hbar, -1, 0, \dots, \phi - 1\}$.

Table 1 The registers maintained by FFMM and SSMM.

register	values set
$step_v$	$\{\hbar, -1, 0, \dots, \phi - 1\}$
m_v	$\{\perp, \texttt{U}\} \cup N(v)$
$In_v^\sigma(\cdot),\ Out_v^\sigma(\cdot)$	$\{\perp, \texttt{U}, \texttt{H}, \texttt{E}\}$
$In_v^r(\cdot),\ Out_v^r(\cdot)$	$\{\perp, \texttt{Acc}, \texttt{Mreq}\}$
$In_v^s(\cdot),\ Out_v^s(\cdot)$	$\{\hbar, -1, 0, \dots, \phi - 1\}$
$active_v$	$\{true, false\}$
$chosen_v$	$\{\perp\} \cup N(v)$
$P(v)$	$2^{N(v)}$