Design of Self-Stabilizing Approximation Algorithms via a Primal-Dual Approach

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

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

Noga Harlev
Technion – Israel Institute of Technology, Haifa, Israel

Abstract

Self-stabilization is an important concept in the realm of fault-tolerant distributed computing. In this paper, we propose a new approach that relies on the properties of linear programming duality to obtain self-stabilizing approximation algorithms for distributed graph optimization problems. The power of this new approach is demonstrated by the following results:

- A self-stabilizing $2(1+\varepsilon)$-approximation algorithm for minimum weight vertex cover that converges in $O(\log \Delta / (\varepsilon \log \log \Delta))$ synchronous rounds.
- A self-stabilizing Δ-approximation algorithm for maximum weight independent set that converges in $O(\Delta + \log^* n)$ synchronous rounds.
- A self-stabilizing $((2\rho + 1)(1+\varepsilon))$-approximation algorithm for minimum weight dominating set in ρ-arboricity graphs that converges in $O((\log \Delta)/\varepsilon)$ synchronous rounds.

In all of the above, Δ denotes the maximum degree. Our technique improves upon previous results in terms of time complexity while incurring only an additive $O(\log n)$ overhead to the message size. In addition, to the best of our knowledge, we provide the first self-stabilizing algorithms for the weighted versions of minimum vertex cover and maximum independent set.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Distributed algorithms

Keywords and phrases self-stabilization, approximation algorithms, primal-dual

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2022.27

Funding This research was supported by VATAT Fund to the Technion Artificial Intelligence Hub (Tech.AI).

Acknowledgements We thank Laurent Feuilloley for a helpful and insightful discussion.

1 Introduction

Distributed networks have become ubiquitous in modern engineering reality. One of the major challenges that arise when dealing with large-scale systems is handling fault recovery. The notion of *self-stabilization* was introduced by Dijkstra [10] to accommodate this challenge. Self-stabilization is characterized by the ability of a distributed system that starts from an arbitrary state to converge into a correct state within a finite time. The initial arbitrary state of the system can capture any finite number of faults, thus making self-stabilization an adaptable fault-tolerance approach.

In the realm of *distributed computing*, classic optimization problems continue to draw much research attention, and new distributed approximation algorithms are always in demand. While an abundance of recent studies have been dedicated to distributed approximation algorithms [3, 4, 11, 14, 23], most of them operate in a *fault-free* environment, i.e., they are assumed to start from some designated initial state.

As a step toward bridging the gap between distributed optimization problems and self-stabilization, in this paper we introduce a new general technique that facilitates the design of self-stabilizing approximation algorithms. We consider distributed algorithms that work in the synchronous *message passing* model. Our technique is based on the *primal-dual* methodology, which is known to be highly useful in the context of approximation algorithms [32]. Given a fault-free approximation algorithm, the technique converts it into a self-stabilizing algorithm with the same approximation and runtime guarantees. Moreover, the conversion induces only an additive $O(\log n)$ overhead to the message size, where n is the number of nodes in the graph. Since the fault-free algorithms used in the context of this paper have a message size of $O(\log n)$ (under common assumptions), we get that all of the self-stabilizing algorithms developed in this paper also have a message size of $O(\log n)$.

In Section 4, we demonstrate the power of our new technique by applying it to three recent fault-free algorithms. This leads to new self-stabilizing approximation algorithms for minimum weight vertex cover, maximum weight independent set, and minimum weight dominating set. To the best of our knowledge, these are the first self-stabilizing algorithms for the weighted versions of minimum vertex cover and maximum independent set, and the first sub-linear time algorithm for minimum weight dominated set.

1.1 Model

Consider an undirected graph $G = (V, E)$ and denote $n = |V|$ and $m = |E|$. For a node $v \in V$, we stick to the convention that $N_G(v)$ denotes the set of v's neighbors in G and that $\deg_G(v)$ denotes v's degree in G. When G is clear from context, we may omit it from our notation and use $N(v)$ and $\deg(v)$ instead of $N_G(v)$ and $\deg_G(v)$, respectively. Let $E(v) = \{e \in E : v \in e\}$ denote the set of edges in E incident on node $v \in V$.

Following a common convention in the realm of *distributed graph algorithms*, additional input components such as node/edge weights and edge orientations, are passed to the nodes of graph G by means of an *input assignment* $\ell : V \to \{0,1\}^*$ which assigns to each node $v \in V$ an *input label* $l(v)$. The input label $\ell(v)$ encodes graph attributes relating to v and its incident edges. Moreover, we assume that $\ell(v)$ includes a *port numbering*, i.e., a bijection between v's incident edges and the set $\{1, \dots, \deg(v)\}$ of ports. Unless stated otherwise, when we refer to an ordered list $u_1, \dots, u_{\deg(v)}$ of v's neighbors, it is assumed that the list is ordered by v's port numbers. We refer to the pair $G_\ell = \langle G, \ell \rangle$ as a *labeled graph*.

In this paper, we focus on algorithms that operate in a *message passing* framework in which the nodes of a given labeled graph G_ℓ are associated with identical state machines that update their state concurrently in synchronous *rounds*. In each round, every node $v \in V$ carries out the following operations: (1) v performs local computation and updates its state as a function of its current state, its input label $\ell(v)$, and possibly random coin tosses; (2) v sends messages to its neighbors; and (3) v receives messages sent to it in the current round by its neighbors. We define the *global state* of G_ℓ to be the n-sized vector encoding the states of all nodes in G.

The state of each node $v \in V$ also includes a designated *output register* $\texttt{out}(v) \in \{0,1\}^* \cup \{\bot\}$ in which v maintains its *output*. If $\texttt{out}(v) = \bot$ we say that v is *undecided*, otherwise, we say that v is *decided*. For a labeled graph G_ℓ, we define a *configuration* of G_ℓ as an n-sized vector $c : V \to \{0,1\}^* \cup \{\bot\}$ assigning an output value $c(v)$ to each node $v \in V$. We refer to the 3-tuple $G_{\ell,c} = \langle G, \ell, c \rangle$ consisting of a graph $G = (V, E)$, an input assignment $\ell : V \to \{0,1\}^*$, and a configuration $c : V \to \{0,1\}^* \cup \{\bot\}$ of G_ℓ, as a *configured graph*.

A *distributed problem* Π is a collection of configured graphs $G_{\ell,c}$. In the context of a distributed problem Π, a labeled graph G_ℓ is said to be *valid* if there exists a configuration c such that $G_{\ell,c} \in \Pi$, in which case we say that c is *feasible* for G_ℓ. Given a distributed problem Π, we may slightly abuse notation and write $G_\ell \in \Pi$ to denote that G_ℓ is valid.

Consider a distributed problem Π. Given a valid labeled graph $G_\ell \in \Pi$, the goal of an algorithm `Alg` for Π is to converge to a feasible configuration within a finite number of rounds, in which case we say that `Alg` is *correct*. When considering an algorithm `Alg` that operates in a *fault-free* environment, the initial state of each node $v \in V$ is assumed to be determined locally by `Alg`. More formally, for each valid labeled graph $G_\ell \in \Pi$, the initial state of each node $v \in V$ is defined to be the value $\texttt{init}_{\texttt{Alg}}(\ell(v))$ obtained by a function $\texttt{init}_{\texttt{Alg}} : \{0,1\}^* \to \{0,1\}^*$. In contrast, *self-stabilizing* algorithms do not determine the initial state of the nodes. That is, we say that an algorithm `Alg` for Π is self-stabilizing if for any valid labeled graph $G_\ell \in \Pi$, algorithm `Alg` is guaranteed to converge to a feasible configuration starting from any initial global state. The *runtime* of an algorithm is defined to be the number of rounds required until convergence.

For many distributed problems, the quality of a feasible configuration can be measured by means of an *objective function* that one wishes to minimize/maximize. Formally, we define a *distributed minimization problem* (resp., *distributed maximization problem*) Ψ as a pair $\langle \Pi, f \rangle$, where Π is a distributed problem, and $f : \Pi \to \mathbb{R}$ is an objective function that assigns an objective value $f(G_{\ell,c})$ to any configured graph $G_{\ell,c} \in \Pi$. For an approximation parameter $\alpha \geq 1$, we say that a configuration c is an *α-approximation* for a valid labeled graph $G_\ell \in \Pi$ if the following conditions hold: (1) $G_{\ell,c} \in \Pi$, i.e., c is feasible (with respect to Π) for G_ℓ; and (2) $f(G_{\ell,c}) \leq \alpha \cdot f(G_{\ell,c'})$ (resp., $f(G_{\ell,c}) \geq f(G_{\ell,c'})/\alpha$) for any feasible configuration c'. We often use the general term *distributed optimization problem* to refer to distributed minimization problems as well as distributed maximization problems. We say that an algorithm `Alg` *α-approximates* a distributed optimization problem Ψ if it solves the distributed problem $\Pi_{\Psi,\alpha} = \{G_{\ell,c} \mid c \text{ is an } \alpha\text{-approximation for } G_\ell\}$.

1.2 Related Work

The notion of self-stabilization was introduced in the seminal paper of Dijkstra [10] and is studied extensively since then. Special interest is given to self-stabilizing graph algorithms, which have natural applications in distributed systems. Awerbuch and Varghese [2] provided a compiler that transforms deterministic synchronous distributed algorithms into self-stabilizing algorithms with the same running time. Note, however, that this held only under the *LOCAL* model and the size of the node states may be unbounded. See [26] for more details on this compiler.

For the unweighted vertex cover problem, a 2-approximation can be achieved by finding a maximal matching. Hsu and Huang [21] presented a self-stabilizing maximal matching algorithm in the *shared memory* model with running time $O(n^3)$, where n is the number of nodes in the graph. Later, this algorithm was reanalyzed to show that its running time is up-bounded by $O(n^2)$ [29], and then an $O(m+n)$ was shown by [20], where m is the number of edges in the graph. The algorithm of Hsu and Haung assumes *sequential adversary*, which means that exactly one node is scheduled for execution at each round. Gradinariu and Tixeuil [17] provided a general scheme to transform an algorithm under a sequential adversary into an algorithm that works under a *distributed adversary*, which selects a subset of the nodes to be executed at each round. Combined with the algorithm of Hsu and Huang, this scheme yields a time complexity of $O(\Delta m)$, where Δ is the maximum degree of the graph. Chattopadhyay et al. [7] and Manne et al. [27] gave self-stabilizing algorithms for maximal matching with quadratic runtime in more general models. Cohen et al. [9] proposed a randomized self-stabilizing algorithm for computing a maximal matching with a time complexity of $O(n^2)$ rounds with high probability.

Kiniwa [24] devised a self-stabilizing vertex cover algorithm that achieves a $(2 - 1/\Delta)$-approximation. This algorithm, which works in the shared memory model, is the first with an approximation ratio less than 2. Turau and Hauck [31] presented a self-stabilizing vertex cover algorithm that computes a $(3 - 2/(\Delta + 1))$-approximation and stabilizes in $O(n + m)$ rounds.

For the minimal dominating set (MDS) problem, Hedetniemi et al. [19] presented a self-stabilizing algorithm under a sequential adversary with a time complexity of $O(n^2)$. Xu et al. [34] proposed a synchronous MDS self-stabilizing algorithm that converges in $O(n)$ rounds. Self-stabilizing MDS algorithms with a linear time complexity under a distributed adversary are presented in [30, 15, 8]. For the minimum weight dominating set (MWDS) problem, Wang et al. [33] were the first to propose a self-stabilizing algorithm that works for general graphs. Their algorithm converges in $O(n^2)$ rounds under a sequential adversary.

For the unweighted MaxIS problem, one can obtain a Δ-approximation by finding a *maximal independent set (MIS)*. The first self-stabilizing algorithm for the MIS problem was introduced by Shukla et al. [28]. Their algorithm converges in $O(n)$ rounds under a sequential adversary. Under a distributed adversary, Ikeda et al. [22] provided an algorithm that converges in $O(n^2)$ rounds, and Goddard et al. [16] proposed a synchronous algorithm that converges in $O(n)$ rounds. Later, Turau [30] designed the first linear time asynchronous MIS algorithm assuming a distributed adversary. Recently, an improved self-stabilizing linear-time asynchronous MIS algorithm was suggested by Arapoglu and Dagdeviren [1], assuming a distributed adversary as well. Blair and Manne [6] suggested a generic mapping from sequential tree algorithms to self-stabilizing tree algorithms. Among other algorithms, this mapping yields a MaxIS algorithm that requires $O(n^2)$ rounds under the *read-write atomicity* assumption.

We refer the interested reader to [18, 12] for extensive surveys on self-stabilizing algorithms and the different models.

2 Preliminaries

Linear Programming and Duality. A *linear program (LP)* consists of a linear objective function to be optimized (i.e., minimized or maximized) subject to linear inequality constraints. Formally, a minimization (resp., maximization) LP is $\min\{\mathbf{c}^{\mathrm{T}}\mathbf{x} \mid \mathbf{A}\mathbf{x} \geq \mathbf{b} \wedge \mathbf{x} \geq \mathbf{0}\}$ (resp., $\max\{\mathbf{c}^{\mathrm{T}}\mathbf{x} \mid \mathbf{A}\mathbf{x} \leq \mathbf{b} \wedge \mathbf{x} \geq \mathbf{0}\}$), where $\mathbf{x} = \{x_j\} \in \mathbb{R}^s$ is a vector of variables and $\mathbf{A} = \{a_{i,j}\} \in \mathbb{R}^{r \times s}$, $\mathbf{b} = \{b_i\} \in \mathbb{R}^r$, and $\mathbf{c} = \{c_j\} \in \mathbb{R}^s$ are a matrix and vectors of coefficients, respectively. An *integer linear program (ILP)* is an LP with integer variables. An *LP relaxation* of an ILP is the LP obtained from the ILP by relaxing its integrality constraints.

Every LP has a corresponding *dual program*, and in this context, we refer to the original LP as the *primal program*. Specifically, for a minimization (resp., maximization) LP, its dual program is a maximization (resp., minimization) LP, formulated as $\max\{\mathbf{b}^{\mathrm{T}}\mathbf{y} \mid \mathbf{A}^{\mathrm{T}}\mathbf{y} \leq \mathbf{c} \wedge \mathbf{y} \geq \mathbf{0}\}$ (resp., $\min\{\mathbf{b}^{\mathrm{T}}\mathbf{y} \mid \mathbf{A}^{\mathrm{T}}\mathbf{y} \geq \mathbf{c} \wedge \mathbf{y} \geq \mathbf{0}\}$). The following properties of LP duality make it a powerful tool. The *weak duality* theorem states that $\mathbf{c}^{\mathrm{T}}\mathbf{x} \geq \mathbf{b}^{\mathrm{T}}\mathbf{y}$ (resp., $\mathbf{c}^{\mathrm{T}}\mathbf{x} \leq \mathbf{b}^{\mathrm{T}}\mathbf{y}$) for every two feasible solutions $\mathbf{x}$ and $\mathbf{y}$ to the primal and dual programs, respectively. The *strong duality* theorem states that $\mathbf{c}^{\mathrm{T}}\mathbf{x} = \mathbf{b}^{\mathrm{T}}\mathbf{y}$ if and only if $\mathbf{x}$ and $\mathbf{y}$ are optimal primal and dual solutions, respectively. The *relaxed complementary slackness* conditions are stated as follows, for given parameters $\beta, \gamma \geq 1$.

- Primal relaxed complementary slackness:
  For every primal variable x_j, if $x_j > 0$, then $c_j/\beta \leq \sum_{i=1}^{r} a_{ij}y_i \leq c_j$ (resp., $c_j \leq \sum_{i=1}^{r} a_{ij}y_i \leq \beta \cdot c_j$).
- Dual relaxed complementary slackness:
  For every dual variable y_i, if $y_i > 0$, then $b_i \leq \sum_{j=1}^{s} a_{ij}x_j \leq \gamma \cdot b_i$ (resp., $b_i/\gamma \leq \sum_{j=1}^{s} a_{ij}x_j \leq b_i$).

If the (primal and dual) relaxed complementary slackness conditions hold, then it is guaranteed that $\mathbf{c}^{\mathrm{T}}\mathbf{x} \leq \beta \cdot \gamma \cdot \mathbf{b}^{\mathrm{T}}\mathbf{y}$ (resp., $\mathbf{c}^{\mathrm{T}}\mathbf{x} \geq \frac{1}{\beta \cdot \gamma} \cdot \mathbf{b}^{\mathrm{T}}\mathbf{y}$). Combined with the weak duality theorem, this means that $\mathbf{x}$ approximates an optimal primal solution by a multiplicative factor of $\beta \cdot \gamma$.

3 Our Technique

In this section, we present a high-level description of our technique for designing self-stabilizing approximation algorithms for a large family of distributed graph optimization problems (henceforth, OptDGPs). We say that an OptDGP Ψ is a *covering* (resp., *packing*) problem if it can be formulated as a minimization (resp., maximization) LP P with a dual LP D such that the variables and constraints of P and D are associated with the nodes and/or edges of the graph.[1] We focus on covering/packing problems that are *locally-constrained* in the sense that a primal/dual constraint associated with a node $v \in V$ or an edge $e \in E$, only involves variables associated with incident nodes and/or edges.

Consider a locally-constrained covering/packing problem Ψ. The technique augments a fault-free distributed α-approximation algorithm `Alg` for Ψ with a *local-checking* procedure, resulting in a self-stabilizing distributed α-approximation algorithm $\texttt{Alg}^{\text{stab}}$. We typically consider a fault-free algorithm `Alg` that admits the following structure: (1) `Alg` maintains a feasible dual solution $\mathbf{y}$ throughout its execution; (2) `Alg` constructs a primal solution $\mathbf{x}$ such that no primal constraint is violated; and (3) throughout its execution, `Alg` maintains the property that $\mathbf{x}$ and $\mathbf{y}$ are not "too far" from each other (e.g., by maintaining relaxed complementary slackness conditions).

We now describe the key ideas behind the transformation of `Alg` into a self-stabilizing algorithm $\texttt{Alg}^{\text{stab}}$. At the heart of this transformation, we have the aforementioned local-checking procedure that is invoked repeatedly at the beginning of each round. The local-checking procedure starts from a *detection* step whose goal is to verify the primal and dual feasibility as well as the approximation guarantees of `Alg`. To that end, during $\texttt{Alg}^{\text{stab}}$ each node $v \in V$ keeps track of all the primal and dual variables that appear in the constraints associated with v and its incident edges $E(v)$. We emphasize that this allows each node to perform the detection step locally without communication.

Following the detection step, $\texttt{Alg}^{\text{stab}}$ branches into one of two possibilities: if the current primal and dual assignments satisfy the detection conditions for a node v, then v proceeds to perform local computation and send messages according to `Alg`; otherwise, v performs a *correction* step. While the details of the correction step are often problem-specific, its common idea is to change the primal and dual variables so that they meet the detection conditions. To preserve consistency between two neighbors v and u regarding their mutual

[1] For simplicity, we assume that each node $v \in V$ knows all of the coefficients of P (and D) that are associated with its neighbors and incident edges (this includes, e.g., node/edge weights, capacities, etc.). We note that this assumption is w.l.o.g. since it can be implemented by means of sending this information through messages at the cost of at most 1 round of communication.

primal and dual variables (i.e., the variables maintained by both v and u), v and u inform each other of the current values assigned to those variables at each round. Following that, upon receiving each other's messages, u and v consistently update their mutual variables (the details of this update are also problem-specific).

Notice that once $\texttt{Alg}^{\text{stab}}$ reaches primal and dual assignments that satisfy the detection conditions for every node $v \in V$, it proceeds to construct the primal and dual solution strictly according to $\texttt{Alg}$. By the correctness of $\texttt{Alg}$, reaching such assignments guarantees that $\texttt{Alg}^{\text{stab}}$ converges to an α-approximation for the OptDGP Ψ. Therefore, the main challenge of $\texttt{Alg}^{\text{stab}}$ is to recover from arbitrary primal and dual assignments to primal and dual assignments that satisfy the detection conditions. As we show in Section 4, for some classical covering/packing problems this recovery process can be obtained using only $O(1)$ rounds. Thus, for those problems, $\texttt{Alg}^{\text{stab}}$ achieves the same (asymptotic) runtime guarantee as $\texttt{Alg}$. Moreover, if we stick to the common assumption that all the primal and dual coefficients of the problems mentioned in Section 4 can be represented using $O(\log n)$ bits, then we get that $s_{\text{stab}} = s_{\texttt{Alg}} + O(\log n)$, where s_{stab} and $s_{\texttt{Alg}}$ denote the message size of $\texttt{Alg}^{\text{stab}}$ and $\texttt{Alg}$, respectively.

4 Results

4.1 Minimum Weight Vertex Cover

Consider a graph $G = (V, E)$ associated with a node-weight function $w : V \to \mathbb{R}_{\geq 0}$. A *vertex cover* is a set $U \subseteq V$ of nodes such that each edge $e \in E$ has at least one endpoint in U. A *minimum weight vertex cover (MWVC)* is a vertex cover U that minimizes $w(U) = \sum_{u \in U} w(u)$. In a natural LP formulation of MWVC, each node $v \in V$ is associated with a variable x_v and each edge $(u, v) \in E$ is associated with a covering constraint $x_u + x_v \geq 1$. In the dual LP, each edge $e \in E$ is associated with a variable y_e and each node $v \in V$ is associated with a packing constraint $\sum_{e \in E(v)} y_e \leq w(v)$.

In this section, we devise a self-stabilizing $2(1 + \varepsilon)$-approximation algorithm for MWVC. More concretely, we constructively prove the following theorem.

▶ **Theorem 4.1.** *There exists a self-stabilizing algorithm that converges to a $2(1 + \varepsilon)$-approximation for MWVC in $O(\log \Delta / \varepsilon \log \log \Delta)$ rounds.*

Our algorithm involves adapting the (fault-free) algorithm by Bar-Yehuda et al. [4] to an algorithm that works in a primal-dual framework, i.e., an algorithm that constructs primal and dual solutions. We then exploit the properties of valid primal and dual solutions to construct a self-stabilizing algorithm, i.e., an algorithm that is guaranteed to converge to a $2(1 + \varepsilon)$-approximation for MWVC from an arbitrary global state (and in particular, an arbitrary assignment to the primal and dual variables). Refer to Pseudocode 1 for the full description of the algorithm. We now give a high-level overview of the algorithm.

Overview of the algorithm. Throughout the execution of Algorithm 1, each node $v \in V$ maintains a primal variable $v.x_v \in \{0, 1, \perp\}$ associated with v, where $v.x_v = \perp$ reflects that v is undecided; $v.x_v = 0$ reflects that v is not in the cover; and $v.x_v = 1$ reflects that v is in the cover. Additionally, for each neighbor $u \in N(v)$, v maintains a primal variable $v.x_u \in \{0, 1, \perp\}$ associated with u and a dual variable $v.y_{u,v} \in \mathbb{R}_{\geq 0}$ associated with the edge (u, v). Each node $v \in V$ also maintains the set $N^{und}(v)$ consisting of v's currently undecided neighbors (according to the $v.x_u$ values) and the value $d(v) = |N^{und}(v)|$. For each neighbor $u \in N(v)$, v also maintains a value $v.d(u)$.

Let us now describe how v operates during a round of Algorithm 1. First, v splits the weight $w(v)$ to $\texttt{threshold}(v) = w(v)/(1+\varepsilon)$ and $\texttt{slack}(v) = w(v) - \texttt{threshold}(v)$. Then, v performs detection, i.e., it checks whether the current assignment to its variables is faulty, and performs correction if necessary. Specifically, v checks the following conditions in order: (1) primal feasibility, i.e., if $v.x_v = 0$, then v checks that $v.x_u = 1$ for all $u \in N(v)$; (2) dual feasibility, i.e., v checks that $\sum_{e \in E(v)} v.y_e \leq w(v)$; and (3) primal relaxed complementary slackness, i.e., if $v.x_v = 1$, then v checks that $\sum_{e \in E(v)} v.y_e \geq \texttt{threshold}(v)$. If conditions (1) or (3) fail, then v sets $v.x_v = \perp$; if condition (2) fails, then v sets $v.y_e = 0$ for each $e \in E(v)$.

After detection, v computes the message it sends each neighbor $u \in N(v)$. Every message from v to $u \in N(v)$ first indicates whether v is decided or undecided. To preserve consistency of shared values, each message from v to $u \in N(v)$ contains the current values of $v.x_v$,$v.y_{u,v}$, and $d(v)$. Upon receiving values $u.x_u$, $u.y_{u,v}$, and $d(u)$, node v updates its own values by setting $v.x_u = u.x_u$, $v.y_{u,v} = \min\{v.y_{u,v}, u.y_{u,v}\}$, and $v.d(u) = d(u)$.

If v is undecided, then for each undecided neighbor $u \in N^{und}(v)$, in addition to the values $v.x_v$,$v.y_{u,v}$, and $d(v)$ node v sends $v.d(u)$ and a real value $\texttt{budget}(v, u)$. The value $\texttt{budget}(v, u)$ is determined based on an ordering $u_1, \dots u_{d(v)}$ of $N^{und}(v)$ as follows. For each $i \in [d(v)]$, v sets $\texttt{budget}(v, u_i) = \min\{\texttt{slack}(u_i)/v.d(u_i), \texttt{bank}(v) - \sum_{j=1}^{i-1} \texttt{budget}(v, u_j)\}$, where $\texttt{bank}(v) = \texttt{threshold}(v) - \sum_{e \in E(v)} v.y_e$. If v receives a message $\langle \texttt{budget}(u, v), d(u), u.d(v), u.y_{u,v} \rangle$ from a neighbor $u \in N^{und}(v)$ that satisfies $d(u) \leq v.d(u)$ and $d(v) \leq u.d(v)$, then v increments the variable $v.y_{u,v}$ by $\texttt{budget}(u, v) + \texttt{budget}(v, u)$.

Finally, if v is undecided, then it becomes decided at the beginning of the following round in one of the following cases: if $\sum_{e \in E(v)} v.y_e \geq \texttt{threshold}(v)$, then v sets $v.x_v = 1$; otherwise, if $v.x_u = 1$ for every neighbor $u \in N(v)$, then v sets $v.x_v = 0$.

Analysis. We now analyze Algorithm 1. Recall that our goal is to establish that Algorithm 1 converges to a $2(1+\varepsilon)$-approximation for MWVC in $O(\log \Delta/\varepsilon \log \log \Delta)$ rounds starting from any global state. To that end, let us first state the following straightforward observation that holds trivially by the construction of Algorithm 1.

▶ **Observation 4.2.** *At the end of each round of Algorithm 1, it holds that $v.x_v = u.x_v$ and $v.y_{u,v} = u.y_{u,v}$ for every $(u, v) \in E$.*

The goal of the following three claims is to show that Algorithm 1 recovers quickly from any global state. As such, these claims play a major role in proving Theorem 4.1.

▷ Claim 4.3. At the end of each round of Algorithm 1, it holds that $\sum_{e \in E(v)} v.y_e \leq w(v)$ for each node $v \in V$.

Proof. Consider some node $v \in V$ and fix some round $i \geq 1$. Notice that the check in line 8 of Pseudocode 1 guarantees that $\sum_{e \in E(v)} v.y_e \leq w(v)$ right before v receives messages. If $v.x_v \neq \perp$ at the time v receives messages, then v will not increase its $v.y_e$ variables, and thus the claim holds. Now, suppose that $v.x_v = \perp$. Let Y^s and Y^f denote the sum of dual variables $v.y_e$ before and after v updates its dual variables in round i, respectively. Let $N'(v) \subseteq N^{und}(v)$ be the set of neighbors $u \in N^{und}(v)$ that send v a message $\langle \texttt{budget}(u, v), d(u), u.d(v), u.y_{u,v} \rangle$ in the current round such that $d(u) \leq v.d(u)$ and $d(v) \leq u.d(v)$. Notice that $Y^f \leq Y^s + \sum_{u \in N'(v)} \texttt{budget}(u, v) + \sum_{u \in N'(v)} \texttt{budget}(v, u)$. By definition, it holds that $\sum_{u \in N'(v)} \texttt{budget}(v, u) \leq \texttt{bank}(v) = \texttt{threshold}(v) - Y^s$. In addition, by the way $\texttt{budget}(u, v)$ is assigned, and since $u.d(v) \geq d(v)$ for all $u \in N'(v)$, and $|N'(v)| \leq d(v)$, it follows that

$$\sum_{u\in N'(v)} \texttt{budget}(u,v) \le \sum_{u\in N'(v)} \frac{\texttt{slack}(v)}{u.d(v)} \le \sum_{u\in N'(v)} \frac{\texttt{slack}(v)}{d(v)} \le \texttt{slack}(v)\,.$$

Overall, we have $Y^f \le \texttt{threshold}(v) + \texttt{slack}(v) = w(v)$. ◁

▷ Claim 4.4. Let $i \ge 2$. At the end of the i-th round of Algorithm 1 it holds that if $v.x_v = 1$, then $\sum_{e\in E(v)} v.y_e \ge \texttt{threshold}(v)$ for each node $v \in V$.

Proof. Consider some node $v \in V$ during the i-th round for some $i \ge 2$. Notice that the check in line 10 of Pseudocode 1 guarantees that v satisfies $v.x_v = 1 \Rightarrow \sum_{e\in E(v)} v.y_e \ge \texttt{threshold}(v)$ right before v receives messages. If $v.x_v \ne 1$ at that time, then $v.x_v \ne 1$ at the end of round i and the claim is trivial; so, suppose that $v.x_v = 1$ right before receiving messages. This means that by the end of round i, node v sets $v.y_{u,v} = \min\{v.y_{u,v}, u.y_{u,v}\}$ for each $u \in N(v)$. Notice that by Claim 4.3, every neighbor $u \in N(v)$ does not satisfy the condition in line 8 of the i-th round. This means that the value $u.y_{u,v}$ sent to v from neighbor u during round i does not change from the end of round $i-1$. From Observation 4.2, it follows that $v.y_{u,v}$ does not change by the end of round i. Therefore, the inequality $\sum_{e\in E(v)} v.y_e \ge \texttt{threshold}(v)$ is still satisfied by the end round i. ◁

▷ Claim 4.5. Let $i \ge 3$. At the end of the i-th round of Algorithm 1 it holds that for each node $v \in V$, if $v.x_v = 0$, then $v.x_u = 1$ for every neighbor $u \in N(v)$.

Proof. Consider some node $v \in V$ during the i-th round for some $i \ge 3$. Notice that the check in line 6 of Pseudocode 1 guarantees that if $v.x_v = 0$, then $v.x_u = 1$ for every neighbor $u \in N(v)$ before v receives messages. By Observation 4.2, at the end of round $i-1$ it holds that $u.x_u = v.x_u = 1$ for all $u \in N(v)$. By Claim 4.4, the value of $u.x_u$ remains 1 for every node $u \in N(v)$ during round $i \ge 3$ (since the condition in line 10 is not satisfied). Therefore, if $v.x_v = 0$, then v receives the message $\langle$"$DECIDED$"$, d(u), u.x_u = 1, u.y_{u,v}\rangle$ from each neighbor $u \in N(v)$ in the i-th round, thus the claim holds at the end of round i. ◁

We are now prepared to prove Theorem 4.1.

Proof of Theorem 4.1. We start with the runtime analysis of Algorithm 1. We note that the runtime analysis uses similar arguments as the analysis presented in [4]. Claims 4.4 and 4.5 imply that if node v is decided (i.e., $v.x_v \ne \bot$) in round $i > 3$, then it will not change its decision at any round $i' \ge i$. We now bound the number of rounds until a node $v \in V$ becomes decided.

Fix some $i > 4$ and node $v \in V$, and suppose that $v.x_v = \bot$ in the i-th round. Let $d_i(v)$ be the value of $d(v)$ in round i (after the update in line 2) and let $Y_i(v)$ be the sum of dual variables $\sum_{e\in E(v)} v.y_e$ at the end of round i. We denote by $u.d_i(v)$ the value of $u.d(v)$ at the beginning of round i for each neighbor $u \in N(v)$. Observe that u updates $u.d(v)$ at the end of round $i-1$ according to a message from v, and thus $u.d_i(v) = d_{i-1}(v)$. We also note that nodes that are decided at the beginning of round $i-1$ do not become undecided at any time afterwards. Therefore, it follows that $d_i(v) \le d_{i-1}(v) = u.d_i(v)$.

We now show that for every parameter $z > 0$, it holds that either (1) $d_{i+2}(v) \le d_{i-1}(v)/z$; or (2) $Y_i(v) \ge Y_{i-1}(v) + \texttt{slack}(v)/z$. First, observe that if during the i-th round it holds that $\texttt{budget}(u,u') < \texttt{slack}(u')/u.d_i(u')$ for some undecided node $u \in V$ and neighbor $u' \in N^{und}(u)$, then u sets $u.x_u = 1$ during round $i+1$ and inform its neighbors. Hence, if $d_{i+2}(v) > d_{i-1}(v)/z$, then v updates $Y_i(v)$ according to more than $d_{i-1}(v)/z$ messages with $\texttt{budget}(u, v) = \texttt{slack}(v)/u.d_i(v) = \texttt{slack}(v)/d_{i-1}(v)$ for every undecided neighbor $u \in N^{und}(v)$. It follows that $Y_i(v) \geq Y_{i-1}(v) + (d_{i-1}(v)/z) \cdot (\texttt{slack}(v)/d_{i-1}(v)) = Y_{i-1}(v) + \texttt{slack}(v)/z$. Recall that v becomes decided in round i' if either $d_{i'}(v) = 0$; or $Y_{i'}(v) \geq \texttt{threshold}(v)$. By the above, case (2) can occur in at most $z \cdot w(v)/\texttt{slack}(v)$ rounds until $\sum_{e \in E(v)} y_e \geq \texttt{threshold}(v)$. As case (1) can occur in at most $\log(\deg(v))/\log z$ rounds, it follows that after

$$\begin{aligned} \frac{z \cdot w(v)}{\texttt{slack}(v)} + O\left(\frac{\log(\deg(v))}{\log z}\right) &= \frac{z}{1 - 1/(1+\varepsilon)} + O\left(\frac{\log(\deg(v))}{\log z}\right) \\ &= \frac{z(1+\varepsilon)}{\varepsilon} + O\left(\frac{\log(\deg(v))}{\log z}\right) \end{aligned}$$

rounds v must be decided. Taking $z = \log(\deg(v))/\log\log(\deg(v))$, we get the desired bound of $O(\log(\deg(v))/\varepsilon \log\log(\deg(v)))$ rounds.

As for the correctness, first notice that by Observation 4.2, it holds at convergence that $v.x_v = u.x_v$ and $v.y_{u,v} = u.y_{u,v}$ for each $(u, v) \in E$. Additionally, by the design of Algorithm 1 and by Claims 4.3, 4.4, and 4.5, the variables' values do not change afterwards. Let $\mathbf{x} = \langle x_v \mid v \in V \rangle \in \{0, 1\}^n$ and $\mathbf{y} = \langle y_e \mid e \in E \rangle \in \mathbb{R}^m_{\geq 0}$ be the primal and dual solutions derived from the variables $v.x_v$ and $v.y_e$, respectively. By relaxed complementary slackness, it is sufficient to show that the following conditions are satisfied: (1) $\mathbf{x}$ is a feasible primal solution; (2) $\mathbf{y}$ is a feasible dual solution; (3) $x_v > 0 \Rightarrow \sum_{e \in E(v)} y_e \geq w(v)/(1+\varepsilon)$; and (4) $y_{u,v} > 0 \Rightarrow x_u + x_v \leq 2$. Conditions (1), (2), and (3) follow directly from Claims 4.5,4.3, and 4.4, respectively. Condition (4) holds trivially, since $x_u + x_v \leq 1 + 1 = 2$ for all $(u, v) \in E$. ◀

Message size. Note that the size of messages sent during Algorithm 1 depends on the values of $\texttt{budget}(v, u)$ computed during its execution. Observe that this dependency is manifested in the $\texttt{budget}(v, u)$ values themselves as well as the dual values $v.y_{u,v}$. As remarked in [4], each $\texttt{budget}(v, u)$ value can be modified to be represented using $O(\log n)$ bits without affecting the correctness or the (asymptotic) runtime of the algorithm. The idea is to round each $\texttt{budget}(v, u)$ value and reduce the $\texttt{slack}(v)$ values accordingly. We note that applying a similar modification to Algorithm 1 is straightforward. Using this modification, we get messages of size $O(\log n)$.

4.2 Maximum Weight Independent Set

Consider a graph $G = (V, E)$ associated with a node-weight function $w : V \to \mathbb{R}_{\geq 0}$. An *independent set* is a set $X \subseteq V$ of nodes such that each edge $e \in E$ has at most one endpoint in X. A *maximum weight independent set (MWIS)* is an independent set $X \subseteq V$ that maximizes $w(X) = \sum_{v \in X} w(v)$. In a natural LP formulation of MWIS, each node $v \in V$ is associated with a variable x_v and each edge $(u, v) \in E$ is associated with a packing constraint $x_u + x_v \leq 1$. In the dual LP, each edge $e \in E$ is associated with a variable y_e and each node $v \in V$ is associated with a covering constraint $\sum_{e \in E(v)} y_e \geq w(v)$.

In this section, we present a self-stabilizing algorithm that given a proper $(\Delta+1)$-coloring, obtains a Δ-approximation for MWIS. More concretely, we constructively prove the following lemma.

▶ **Lemma 4.6.** *Given a proper $(\Delta + 1)$-coloring $c : V \to \{1, \ldots, \Delta + 1\}$, there exists a self-stabilizing algorithm that converges to a Δ-approximation for MWIS in $O(\Delta)$ rounds.*

Algorithm 1 A self-stabilizing $2(1+\varepsilon)$-approximation algorithm for MWVC. Code for node $v \in V$ in a single round.

```
1:  $\texttt{threshold}(v) = w(v)/(1+\varepsilon)$; $\texttt{slack}(v) = w(v) - \texttt{threshold}(v)$
2:  $N^{und}(v) = \{u \in N(v) \mid v.x_u = \perp\}$; $d(v) = |N^{und}(v)|$          ▷ $v$'s undecided neighbors
3:  if $v.x_v == \perp$ then
4:      if $\sum_{e \in E(v)} v.y_e \geq \texttt{threshold}(v)$ then $v.x_v = 1$
5:      else if $v.x_u == 1$ for every neighbor $u \in N(v)$ then $v.x_v = 0$
6:  if $(v.x_v == 0) \wedge (\exists u \in N(v) : v.x_u \neq 1)$ then          ▷ checking primal feasibility
7:      $v.x_v = \perp$
8:  if $\sum_{e \in E(v)} v.y_e > w(v)$ then          ▷ checking dual feasibility
9:      $v.y_e = 0$ for all $e \in E(v)$
10: if $(v.x_v == 1) \wedge (\sum_{e \in E(v)} v.y_e < \texttt{threshold}(v))$ then          ▷ checking comp. slackness
11:     $v.x_v = \perp$
12: if $v.x_v \in \{0, 1\}$ then
13:     send $\langle$"$DECIDED$"$, d(v), v.x_v, v.y_{u,v}\rangle$ to each neighbor $u \in N(v)$
14: else
15:     send $\langle$"$UNDECIDED$"$, d(v), v.y_{u,v}\rangle$ to each neighbor $u \in N(v) - N^{und}(v)$
16:     $\texttt{bank}(v) = \texttt{threshold}(v) - \sum_{e \in E(v)} v.y_e$
17:     let $u_1, \ldots, u_{d(v)}$ be an ordering of $N^{und}(v)$          ▷ e.g., by port numbers
18:     for $i = 1, \ldots, d(v)$ do
19:         $\texttt{slack}(u_i) = (1 - 1/(1+\varepsilon)) \cdot w(u_i)$
20:         $\texttt{budget}(v, u_i) = \min\{\texttt{slack}(u_i)/v.d(u_i), \texttt{bank}(v) - \sum_{j=1}^{i-1} \texttt{budget}(v, u_j)\}$
21:         send $\langle \texttt{budget}(v, u_i), d(v), v.d(u), v.y_{u_i,v}\rangle$ to $u_i$
22: for each message $\mu_u$ received from neighbor $u \in N(v)$ do
23:     if $\mu_u == \langle$"$DECIDED$"$, d(u), u.x_u, u.y_{u,v}\rangle$ then
24:         $v.d(u) = d(u)$; $v.x_u = u.x_u$; $v.y_{u,v} = \min\{v.y_{u,v}, u.y_{u,v}\}$
25:     if $\mu_u == \langle$"$UNDECIDED$"$, d(u), u.y_{u,v}\rangle$ then
26:         $v.d(u) = d(u)$; $v.x_u = \perp$; $v.y_{u,v} = \min\{v.y_{u,v}, u.y_{u,v}\}$
27:     if $\mu_u == \langle \texttt{budget}(u, v), d(u), u.d(v), u.y_{u,v}\rangle$ then
28:         $v.x_u = \perp$; $v.y_{u,v} = \min\{v.y_{u,v}, u.y_{u,v}\}$
29:         if $(u \in N^{und}(v)) \wedge (v.x_v == \perp) \wedge (d(u) \leq v.d(u)) \wedge (d(v) \leq u.d(v))$ then
30:             $v.y_{u,v} = v.y_{u,v} + \texttt{budget}(u, v) + \texttt{budget}(v, u)$
31:         $v.d(u) = d(u)$
```

Our algorithm involves adapting the (fault-free) algorithm by Bar-Yehuda et al. [3] to a primal-dual algorithm, and then applying our technique to obtain a self-stabilizing algorithm. Refer to Pseudocode 2 for a full description of the algorithm. For simplicity of presentation, we assume that each node knows the colors of all its neighbors.

Combining Lemma 4.6 with the self-stabilizing $(\Delta + 1)$-coloring algorithm by Barenboim et al. [5] (henceforth referred to as the BEG algorithm), we establish the following theorem.[2]

[2] We note that the BEG algorithm requires that the nodes' labels include the values of Δ and n, as well as a unique ID.

▶ **Theorem 4.7.** *There exists a self-stabilizing algorithm that converges to a Δ-approximation for MWIS in $O(\Delta + \log^* n)$ rounds.*

We remark that incorporating the BEG algorithm into Algorithm 2 can be done in a straightforward manner. This requires the nodes to repeatedly check that the current coloring is proper, and correct it according to the BEG algorithm if necessary. As established in [5], the BEG algorithm converges to a proper $(\Delta + 1)$-coloring in $O(\Delta + \log^* n)$ rounds. The execution of the BEG algorithm is performed in parallel to Algorithm 2 so that after $O(\Delta + \log^* n)$ rounds, the incorporated algorithm performs its updates strictly based on Algorithm 2.

Overview of Algorithm 2. Throughout the execution of Algorithm 2, each neighbor $v \in V$ maintains a primal variable $v.x_v \in \{0, 1\}$ and a dual variable $v.y_{u,v} \in \mathbb{R}_{\geq 0}$ for each node $u \in N(v)$.[3] Additionally, v maintains a primal variable $v.x_u$ for each neighbor $u \in N(v)$.

Consider a node $v \in V$ and let $S(v) = \{u \in N(v) \mid c(u) < c(v)\}$ and $L(v) = N(v) - S(v)$. At each round, v updates its primal and dual variables as follows. If $\sum_{u \in S(v)} v.y_{u,v} \geq w(v)$, then v sets $v.x_v = 0$ and $v.y_{u,v} = 0$ for each $u \in L(v)$. Otherwise, v sets $v.y_{u,v} = w(v) - \sum_{u' \in S(v)} v.y_{u',v}$ for each $u \in L(v)$. In the case that $\sum_{u \in S(v)} v.y_{u,v} < w(v)$, node v sets $v.x_v = 0$ if there exists a neighbor $u \in L(v)$ such that $v.x_u = 1$. Otherwise, node v sets $v.x_v = 1$.

At the end of each round, v sends the dual variable $v.y_{u,v}$ to each neighbor $u \in L(v)$, and the primal variable $v.x_v$ to each neighbor $u \in S(v)$. Upon receiving a message $u.y_{u,v}$ (resp., $u.x_u$) from a neighbor $u \in S(v)$ (resp., $u \in L(v)$), node v sets $v.y_{u,v} = u.y_{u,v}$ (resp., $v.x_u = u.x_u$).

Analysis. We now turn to analyze Algorithm 2. To that end, let us first state the following straightforward observation that holds trivially by the construction of Algorithm 2.

▶ **Observation 4.8.** *Consider a node $v \in V$. At the end of each round of Algorithm 2, it holds that $v.x_u = u.x_u$ for each neighbor $u \in L(v)$; and $v.y_{u,v} = u.y_{u,v}$ for each neighbor $u \in S(v)$.*

The following two claims are used to establish the convergence time of the primal and dual solutions.

▷ Claim 4.9. Fix some node $v \in V$. During the execution of Algorithm 2, the dual variable $v.y_{u,v}$ does not change at any time from the end of round $c(v)$ for every $u \in N(v)$.

Proof. We prove the claim by induction on $c(v) = 1, \ldots, \Delta + 1$. For the base of the induction, consider the case where $c(v) = 1$. This means that $S(v) = \emptyset$ and thus v sets $v.y_{u,v} = w(v) - \sum_{u' \in S(v)} v.y_{u',v} = w(v)$ at round 1 for each $u \in N(v)$. By the construction of Algorithm 2, these values do not change afterwards.

Now, suppose that $i = c(v) > 1$. Notice that by Observation 4.8, it holds that $v.y_{u,v} = u.y_{u,v}$ for each $u \in S(v)$ at the beginning of round i. By the induction hypothesis, these variables do not change throughout the execution from round i onward. Notice that for each neighbor $u \in L(v)$, node v sets $v.y_{u,v} = 0$ in the case that $\sum_{u \in S(v)} v.y_{u,v} \geq w(v)$; and $v.y_{u,v} = w(v) - \sum_{u' \in S(v)} v.y_{u',v}$ otherwise. Since the value $v.y_{u',v}$ does not change for each

[3] We note that unlike the other algorithms presented in this paper, the dual solution obtained by the dual variables in Algorithm 2 is not necessarily feasible. We elaborate on that in the proof of Lemma 4.6.

node $u' \in S(v)$, it follows that the value $v.y_{u,v}$ does not change for every $u \in L(v)$. Overall, we conclude that the value $v.y_{u,v}$ does not change from the end of round $c(v)$ onward for every $u \in S(v) \cup L(v) = N(v)$. $\lhd$

$\rhd$ Claim 4.10. Fix some node $v \in V$. During the execution of Algorithm 2, the primal variable $v.x_v$ does not change at any time from the end of round $2\Delta + 3 - c(v)$.

Proof. We prove the claim by induction on $c(v) = \Delta + 1, \ldots, 1$. For the base of the induction, suppose that $c(v) = \Delta + 1$ and consider round $\Delta + 2 = 2\Delta + 3 - c(v)$. Since $c(v) = \Delta + 1$, it follows that $L(v) = \emptyset$ and v sets $v.x_v = 0$ if $\sum_{u \in S(v)} v.y_{u,v} \geq w(v)$; and $v.x_v = 1$ otherwise. By Claim 4.9, the value $v.y_{u,v}$ for each neighbor $u \in S(v)$ does not change throughout the execution from the end of round $\Delta + 1$. Therefore, it follows that the value $v.x_v$ does not change from the end of round $\Delta + 2$ onward.

Let $v \in V$ such that $c(v) < \Delta + 1$, and consider round $i = 2\Delta + 3 - c(v)$. If at the beginning of round i it holds that $\sum_{u \in S(v)} v.y_{u,v} \geq w(v)$, then v sets $v.x_v = 0$. By Claim 4.9, the value of $\sum_{u \in S(v)} v.y_{u,v}$ does not change after round i and thus it follows that $v.x_v = 0$ at all times from round i onward. Now, suppose that $\sum_{u \in S(v)} v.y_{u,v} < w(v)$. Notice that v sets $v.x_v = 0$ if there exists a neighbor $u \in L(v)$ such that $v.x_u = 1$; and $v.x_v = 1$ otherwise. By Observation 4.8, it holds that $v.x_u = u.x_u$ for each $u \in L(v)$ at the beginning of round i. By the induction hypothesis, these variables do not change throughout the execution from round i onward. Hence, the value $v.x_v$ does not change either. $\lhd$

We are now prepared to prove Lemma 4.6.

Proof of Lemma 4.6. From Claims 4.9 and 4.10, we can deduce that Algorithm 2 converges to a primal solution $\mathbf{x} = \langle x_v \mid v \in V \rangle \in \{0,1\}^n$ derived from the variables $v.x_v$ and a dual solution $\mathbf{y} = \langle y_{(u,v)} \mid (u,v) \in E \rangle \in \mathbb{R}^m_{\geq 0}$ derived from the variables $v.y_{u,v}$ after at most $2\Delta + 2$ rounds. Let $\lambda(v) = \max\{0, w(v) - \sum_{u \in S(v)} y_{u,v}\}$ for each node $v \in V$. Notice that $\mathbf{y}$ is constructed such that $y_{u,v} = \lambda(v)$ for each $u \in L(v)$.

Recall the dual constraint $\sum_{u \in N(v)} y_{u,v} \leq w(v)$ associated with each node v. Notice that $\mathbf{y}$ is constructed such that if $\sum_{u \in S(v)} y_{u,v} < w(v)$ for node $v \in V$, then $y_{u,v} = \lambda(v) = w(v) - \sum_{u' \in S(v)} y_{u',v}$ for each $u \in L(v)$. Thus, the dual constraint is violated only for nodes $v \in V$ such that $L(v) = \emptyset$ and $\sum_{u \in S(v)} y_{u,v} < w(v)$. For the sake of the analysis, we fix the dual feasibility by defining the dual solution $\mathbf{y}'$ as follows. If a node v satisfies $L(v) = \emptyset$ and $\sum_{u \in S(v)} y_{u,v} < w(v)$, then we set the dual value $y'_{z,v} = y_{z,v} + \lambda(v)$ for a single neighbor $z \in S(v)$, and set $y'_{u,v} = y_{u,v}$ for every other neighbor $u \in S(v) - \{z\}$. Otherwise (if $L(v) \neq \emptyset$ or $\sum_{u \in S(v)} y_{u,v} \geq w(v)$), we set the dual value $y'_{u,v} = y_{u,v}$ for every neighbor $u \in S(v)$. It is not hard to see that $\mathbf{y}'$ is a feasible dual solution. In addition, notice that $\mathbf{x}$ is a feasible primal solution since for each node $v \in V$, if $x_v = 1$, then $x_u = 0$ for each neighbor $u \in L(v)$.

Let $X = \{v \mid x_v = 1\}$ be the independent set obtained by Algorithm 2 and consider a node $v \in X$. Let $\mu(v) = \{(u,u') \in E \mid u \in S(v) \wedge u' \in L(u) \wedge x_{u'} = 0\}$ and notice that $y_e = y'_e$ for every edge $e \in \mu_v$. We argue that $\Delta \cdot w(v) \geq \sum_{u \in N(v)} y'_{u,v} + \sum_{e \in \mu(v)} y'_e$ for each $v \in X$. To establish that, first suppose that $L(v) \neq \emptyset$. It holds that

$$\begin{aligned}
\sum_{u \in N(v)} y'_{u,v} + \sum_{e \in \mu(v)} y'_e &= \sum_{u \in N(v)} y_{u,v} + \sum_{e \in \mu(v)} y_e = \sum_{u \in L(v)} y_{u,v} + \sum_{u \in S(v)} y_{u,v} + \sum_{e \in \mu(v)} y_e \\
&\leq \sum_{u \in L(v)} y_{u,v} + \sum_{u \in S(v)} \sum_{u' \in L(u)} y_{u,u'} \\
&= |L(v)| \cdot \lambda(v) + \sum_{u \in S(v)} |L(u)| \cdot \lambda(u)
\end{aligned}$$

$$
\begin{aligned}
&\leq \Delta \left(\lambda(v) + \sum_{u \in S(v)} \lambda(u) \right) \\
&= \Delta \left(w(v) - \sum_{u \in S(v)} y_{u,v} + \sum_{u \in S(v)} \lambda(u) \right) \\
&= \Delta \left(w(v) - \sum_{u \in S(v)} \lambda(u) + \sum_{u \in S(v)} \lambda(u) \right) = \Delta \cdot w(v) \,.
\end{aligned}
$$

Now, suppose that $L(v) = \emptyset$. Notice that since $v \in X$, it must hold that $\sum_{u \in S(v)} y_{u,v} < w(v)$. By the definition of $\mathbf{y}'$, it holds that $y'_{z,v} = y_{z,v} + \lambda(v)$ for a single neighbor $z \in S(v) = N(v)$, and $y'_{u,v} = y_{u,v}$ for every other neighbor $u \in S(v) - \{z\}$. It follows that

$$
\begin{aligned}
\sum_{u \in N(v)} y'_{u,v} + \sum_{e \in \mu(v)} y'_e &= \lambda(v) + \sum_{u \in S(v)} y_{u,v} + \sum_{e \in \mu(v)} y_e = w(v) + \sum_{e \in \mu(v)} y_e \\
&\leq w(v) + \sum_{u \in S(v)} \sum_{u' \in L(u) - \{v\}} y_{u,u'} \\
&\leq w(v) + (\Delta - 1) \sum_{u \in S(v)} \lambda(u) \\
&< \Delta \cdot w(v) \,,
\end{aligned}
$$

where the last transition holds because $x_v = 1$ implies that $\sum_{u \in S(v)} \lambda(u) < w(v)$.

Observe that for every node $v \notin X$, if $L(v) \cap X = \emptyset$, then it holds that $\sum_{u \in S(v)} y'_{u,v} = \sum_{u \in S(v)} y_{u,v} \geq w(v)$ and thus $y'_{u',v} = y_{u',v} = 0$ for each $u' \in L(v)$. Therefore, it follows that

$$
\sum_{e \in E} y'_e = \sum_{v \in V} \sum_{u \in L(v)} y'_{u,v} = \sum_{v \in X} \left(\sum_{u \in N(v)} y'_{u,v} + \sum_{e \in \mu(v)} y'_e \right) \leq \sum_{v \in X} \Delta \cdot w(v) = \Delta \cdot w(X)
$$

From the weak duality theorem, we conclude that X is a Δ-approximation for MWIS. ◀

Algorithm 2 A self-stabilizing Δ-approximation algorithm for MWIS given a proper $(\Delta + 1)$-coloring $c : V \to [\Delta + 1]$. Code for node $v \in V$ in a single round.

```
1:  $S(v) = \{u \in N(v) \mid c(u) < c(v)\}$                        ▷ $v$'s neighbors with a smaller color
2:  $L(v) = N(v) - S(v)$                                             ▷ $v$'s neighbors with a larger color
3:  if $\sum_{u \in S(v)} v.y_{u,v} \geq w(v)$ then
4:      $v.x_v = 0$
5:      $v.y_{u,v} = 0, \ \forall u \in L(v)$
6:  else
7:      $v.y_{u,v} = w(v) - \sum_{u' \in S(v)} v.y_{u',v}, \ \forall u \in L(v)$      ▷ $w(v) > \sum_{u' \in S(v)} v.y_{u',v}$
8:      if $\exists u \in L(v) : v.x_u == 1$ then
9:          $v.x_v = 0$
10:     else $v.x_v = 1$
11: send $v.y_{u,v}$ to each neighbor $u \in L(v)$
12: send $v.x_v$ to each neighbor $u \in S(v)$
13: for each $u.y_{u,v}$ received from neighbor $u \in S(v)$ do $v.y_{u,v} = u.y_{u,v}$
14: for each $u.x_u$ received from neighbor $u \in L(v)$ do $v.x_u = u.x_u$
```

4.3 Minimum Weight Dominating Set in Bounded Arboricity Graphs

Consider a graph $G = (V, E)$ associated with a node-weight function $w : V \to \mathbb{R}_{\geq 0}$. Let us denote $N^+(v) = N(v) \cup \{v\}$ for each node v. We naturally extend this notation to node sets and denote $N^+(X) = \bigcup_{v \in X} N^+(v)$ for a node set $X \subseteq V$. A set $X \subseteq V$ of nodes is said to be a dominating set if $N^+(X) = V$. A *minimum weight dominating set (MWDS)* is a dominating set X that minimizes $w(X)$. In a natural LP formulation for MWDS, each node $v \in V$ is associated with a variable x_v and a covering constraint $\sum_{u \in N^+(v)} x_u \geq 1$. In the dual LP, each node $v \in V$ is associated with a variable y_v and a packing constraint $\sum_{u \in N^+(v)} y_u \leq w(v)$.

In this section, we focus on graphs with bounded *arboricity*. The arboricity of graph G is the minimal number ρ for which there exists a partition $E = E_1 \dot\cup, \dots, \dot\cup E_\rho$ such that E_i induces a forest for each $i \in [\rho]$. We obtain the following results for MWDS on graphs with arboricity at most ρ.

▶ **Theorem 4.11.** *There exists a self-stabilizing algorithm that converges to a $((2\rho+1)(1+\varepsilon))$-approximation for MWDS in graphs with arboricity at most ρ in $O(\log \Delta/\varepsilon)$ rounds.*

Our algorithm is based on the primal-dual algorithm by Dory et al. [11]. We assume w.l.o.g. that every node $v \in V$ knows the value $w_{\min}(u) = \min_{u' \in N^+(u)}\{w(u')\}$ of each of its neighbors $u \in N(v)$. We further assume that for each $u \in V$, the neighbor $\arg\min_{u' \in N^+(u)}\{w(u')\}$ is unique (breaking ties, e.g., by port numbers) and that each node v knows if it is the node that realizes $\arg\min_{u' \in N^+(u)}\{w(u')\}$ for each neighbor $u \in N(v)$. Finally, we assume that the arboricity ρ and the maximum degree Δ are encoded in the label of each node $v \in V$. As remarked in [11], the latter assumption can be lifted by replacing Δ with $\max_{u \in N^+(v)}\{\deg(u)\}$ without affecting the correctness and (asymptotic) runtime of the algorithm. Refer to Pseudocode 3 for a full description of the algorithm.

Algorithm 3 A self-stabilizing $((2\rho + 1)(1 + \varepsilon))$-approximation algorithm for MWDS in graphs with arboricity at most ρ. Code for node $v \in V$ in a single round.

```
1: $\lambda = 1/((2\rho + 1)(1 + \varepsilon))$; $reset = FALSE$
2: MWDS_update_variables                          ▷ may change the value of $reset$
3: MWDS_update_status
4: if $reset == TRUE$ then
5:     send $\langle$"$RESET$", status$(v), v.x_v, v.y_v\rangle$ to each neighbor $u \in N(v)$
6: else
7:     if status$(v) == active$ then
8:         $v.y_v = (1 + \varepsilon)v.y_v$
9:     else if status$(v) == waiting$ then
10:        if $\forall u \in N(v) : v.$status$(u) \neq active$ then
11:            $v.y_v = (1 + \varepsilon)v.y_v$
12:            status$(v) = done\_waiting$
13:    send $\langle$status$(v), v.x_v, v.y_v\rangle$ to each $u \in N(v)$
14:    MWDS_receive_messages
```

Overview of the algorithm. Let us first briefly describe the high-level idea of the (fault-free) algorithm presented in [11]. Throughout the execution, the algorithm maintains a feasible dual solution $\mathbf{y}$ and uses it to construct a dominating set X such that at termination, $w(X)$ is within a multiplicative $((2\rho + 1)(1 + \varepsilon))$ factor from the objective value of $\mathbf{y}$. This is done in two stages. In the first stage, the algorithm constructs a set X_1 which consists of nodes

Algorithm 4 Procedure `MWDS_update_variables`. Node v updates its variables.

```
 1: $v.y_v = \max\{v.y_v, w_{\min}(v)/(\Delta+1)\}$    ▷ setting a lower bound for dual variables
 2: if $\exists u \in N^+(v) : v.\mathtt{status}(u) == \mathit{done\_waiting}$ then
 3:     if $v = \arg\min_{z \in N^+(u)}\{w(z)\}$ then    ▷ breaking ties by port numbers
 4:         $v.x_v = 1$
 5: else
 6:     if $\sum_{u \in N^+(v)} v.y_u > w(v)$ then    ▷ checking dual feasibility
 7:         $v.y_u = w_{\min}(u)/(\Delta+1), v.x_u = 0$ for all $u \in N^+(v)$
 8:         $\mathit{reset} = \mathit{TRUE}$
 9:     if $v.y_v \leq \lambda \cdot w_{\min}(v)$ then
10:         if $\sum_{u \in N^+(v)} v.y_u < w(v)/(1+\varepsilon)$ then    ▷ maintaining primal comp. slackness
11:             $v.x_v = 0$
12:         else
13:             $v.x_v = 1$
14:     else if $\exists u \in N(v) : v.\mathtt{status}(u) == \mathit{active}$ then
15:         $v.y_v = \lambda \cdot w_{\min}(v)$
```

Algorithm 5 Procedure `MWDS_update_status`. Node v updates its status.

```
 1: if $v.y_v \leq w_{\min}(v) \cdot \lambda/(1+\varepsilon)$ then
 2:     if $\sum_{u \in N^+(v)} v.x_u == 0$ then    ▷ primal constraint is not satisfied
 3:         $\mathtt{status}(v) = \mathit{active}$
 4:     else
 5:         $\mathtt{status}(v) = \mathit{over}$
 6: else if $v.y_v \leq w_{\min}(v) \cdot \lambda$ then
 7:     if $\sum_{u \in N^+(v)} v.x_u == 0$ then
 8:         $\mathtt{status}(v) = \mathit{waiting}$
 9:     else
10:         $\mathtt{status}(v) = \mathit{over}$
11: else $\mathtt{status}(v) = \mathit{done\_waiting}$
```

Algorithm 6 Procedure `MWDS_receive_messages`. Node v receives messages from its neighbors.

```
1: for each message $\mu_u$ received from neighbor $u \in N(v)$ do
2:     if $\mu_u == \langle$"$\mathit{RESET}$"$, \mathtt{status}(u), u.x_u, u.y_u\rangle$ then
3:         $v.y_v = w_{\min}(v)/(\Delta+1)$; $v.y_u = u.y_u$; $v.x_u = u.x_u$; $v.\mathtt{status}(u) = \mathtt{status}(u)$
4:     else    ▷ $\mu_u = \langle \mathtt{status}(u), u.x_u, u.y_u, u.y_v\rangle$
5:         $v.y_u = u.y_u$; $v.x_u = u.x_u$; $v.\mathtt{status}(u) = \mathtt{status}(u)$
```

$v \in V$ that satisfy the following two conditions by the end of the stage: (1) $y_u \leq \lambda w_{\min}(u)$ for every $u \in N^+(v)$, where $\lambda = 1/((2\rho+1)(1+\varepsilon))$; and (2) $\sum_{u \in N^+(v)} y_u \geq w(v)/(1+\varepsilon)$. In the second stage, a set X_2 is constructed greedily by having each node $u \in V - N^+(X_1)$ which is not dominated by X_1, add to X_2 a node $v \in N^+(u)$ that satisfies $w(v) = w_{\min}(u)$. As shown in [11], the set $X = X_1 \cup X_2$ is a $((2\rho+1)(1+\varepsilon))$-approximation of MWDS.

In Algorithm 3, we modify the algorithm of [11] to produce a self-stabilizing algorithm. The challenge of such algorithm is to recover from an arbitrary primal and dual assignment. To that end, each node $v \in V$ maintains a primal variable $v.x_v \in \{0,1\}$, a dual variable $v.y_v \in \mathbb{R}_{\geq 0}$, and the primal and dual variables $v.x_u$ and $v.y_u$ of each neighbor $u \in N(v)$. In addition, v maintains a variable $\texttt{status}(v) \in \{\mathit{active}, \mathit{over}, \mathit{waiting}, \mathit{done\_waiting}\}$ and the status $v.\texttt{status}(u)$ of each neighbor $u \in N(v)$.

The role of $\texttt{status}(v)$ is to reflect the current stage of node v with respect to the variables of its neighbors. For each node $v \in V$, $\texttt{status}(v) = \mathit{active}$ reflects that $y_v \leq \lambda w_{\min}(v)/(1+\varepsilon)$ and v is currently not dominated; $\texttt{status}(v) = \mathit{waiting}$ reflects that $\lambda w_{\min}(v)/(1+\varepsilon) < y_v \leq \lambda w_{\min}(v)$ and v is currently not dominated; $\texttt{status}(v) = \mathit{over}$ reflects that $y_v \leq \lambda w_{\min}(v)$ and v is dominated; and $\texttt{status}(v) = \mathit{done\_waiting}$ reflects that $y_v > \lambda w_{\min}(v)$.

At the beginning of each round, each node updates its variables using Procedure 4. This procedure makes sure that dual feasibility is maintained, and also updates the primal variables to achieve similar guarantees to those of [11]. In addition, to enable quick convergence of Algorithm 3, the procedure bounds the dual variables from below by setting $v.y_v = \max\{v.y_v, w\_\min(v)/(\Delta+1)\}$.

Following the update of the variables, each node updates its status according to Procedure 5. Then, if the dual constraint of node v was violated in Procedure 4, then v resets the dual variables of its neighbors and informs them by sending a "*RESET*" message. Nodes that receive a "RESET" message set $v.y_v$ to $w_{\min}(v)/(\Delta+1)$.

If the dual constraint was not violated, v increases its dual variable by setting $v.y_v = (1+\varepsilon)v.y_v$ if one of the following cases holds: (1) $\texttt{status}(v) = \mathit{active}$; or (2) $\texttt{status}(v) = \mathit{waiting}$ and $v.\texttt{status}(u) \neq \mathit{active}$ for each $u \in N(v)$. In the latter case, v also sets $\texttt{status}(v) = \mathit{done\_waiting}$. Finally, v informs its neighbors about its status and current values of variables, and uses the messages received to update the values of variables $v.x_u$, $v.y_u$, and $v.\texttt{status}(u)$ of each neighbor $u \in N(v)$ according to Procedure 6.

Analysis. We now turn to analyze Algorithm 3. To that end, let us first state the following straightforward observation that holds trivially by the construction of Algorithm 3.

▶ **Observation 4.12.** *At the end of each round of Algorithm 3, it holds that $v.y_v = u.y_v$, $v.x_v = u.x_v$, and $\texttt{status}(v) = u.\texttt{status}(v)$ for every $(u,v) \in E$.*

We now establish an important property regarding the dual solution maintained by Algorithm 3.

▷ Claim 4.13. Let $i \geq 2$. At the end of the i-th round of Algorithm 3, it holds that $\sum_{u \in N^+(v)} v.y_u \leq w(v)$ for every node $v \in V$.

Proof. Fix some node $v \in V$. First, suppose that $\sum_{u \in N^+(v)} v.y_u \leq w(v)$ at the beginning of round i. By Observation 4.12, at the beginning of round i it holds that $v.y_u = u.y_u$ for every $u \in N(v)$. Notice that v updates the dual variable $v.y_u$ at the end of the round according to the message from neighbor u. Each update increases the dual variable $v.y_u$ by a multiplicative factor of at most $(1+\varepsilon)$. We argue that this update does not violate the inequality in the statement. Notice that if $v.x_v = 0$, then by the construction of Procedure 4 it follows that $\sum_{u\in N^+(v)} v.y_u < w(v)/(1+\varepsilon)$ at the beginning of the round. Therefore, an increase of this sum by a multiplicative $(1+\varepsilon)$ does not exceed $w(v)$. If $v.x_v = 1$, then each neighbor $u \in N(v)$ sets its status to *over* and does not increase $u.y_u$.

Now, suppose that $\sum_{u\in N^+(v)} v.y_u > w(v)$ at the beginning of round i. This means that v sets $v.y_u = w_{\min}(u)/(\Delta+1)$ for each $u \in N^+(v)$. Therefore, at the end of round i it holds that

$$\sum_{u\in N^+(v)} v.y_u \;=\; \sum_{u\in N^+(v)} \frac{w_{\min}(u)}{\Delta+1} \;\leq\; \frac{(\deg(v)+1)\cdot w(v)}{\deg(v)+1} \;=\; w(v)\,,$$

thus establishing the assertion. ◁

From Claim 4.13, we deduce the following corollary.

▶ **Corollary 4.14.** *Consider a node $v \in V$. During the execution of Algorithm 3, the value $v.y_u$ does not decrease from round 3 onward for each $u \in N^+(v)$.*

We can now show the following claim.

▷ Claim 4.15. Let $i \geq 3$ and consider a node $v \in V$. If $v.x_v = 1$ at the end of round i, then $v.x_v = 1$ at each round $i' \geq i$.

Proof. Observe that $v.x_v = 1$ at the end of round i in one of the following cases: (1) there exists a node $u \in N^+(v)$ such that $\texttt{status}(u) = \textit{done\_waiting}$ and $w_{\min}(u) = w(v)$; or (2) node v satisfies the inequality $\sum_{u\in N^+(v)} v.y_u \geq w(v)/(1+\varepsilon)$ at the beginning of the i-th round. In case (1), we note that by Corollary 4.14 node u will not decrease its dual variable throughout the execution of Algorithm 3. This means that $\texttt{status}(u) = \textit{done\_waiting}$ throughout the execution and thus it follows that $v.x_v = 1$. For case (2), by Corollary 4.14 the value of $v.y_u$ does not decrease for all $u \in N^+(v)$ throughout the execution of Algorithm 3. Therefore, the inequality $\sum_{u\in N^+(v)} v.y_u \geq w(v)/(1+\varepsilon)$ remains satisfied. ◁

We conclude the analysis by proving Theorem 4.11.

Proof of Theorem 4.11. First, observe that by Claim 4.15 and the construction of Procedure 4, if $v.x_v = 1$ for node $v \in V$ at some round $i \geq 3$, then $v.x_v = 1$ from round i onward. To see that Algorithm 3 converges to a dominating set, observe that for each node $v \in V$, if $\texttt{status}(v) \in \{\textit{over}, \textit{done\_waiting}\}$ at some round $i \geq 1$, then it follows that there exists a node $u \in N^+(v)$ such that $u.x_u = v.x_u = 1$ by the end of round i. Notice that $v.y_v \geq w_{\min}(u)/(\Delta+1)$ for each $v \in V$ throughout the execution. It now follows from Claim 4.14 and the design of Algorithm 3 that after $O(\log\Delta/\varepsilon)$ rounds, each node $v \in V$ satisfies $\texttt{status}(v) \in \{\textit{over}, \textit{done\_waiting}\}$.

We are now ready to establish the correctness of Algorithm 3. Observe that if $\texttt{status}(v) \in \{\textit{over}, \textit{done\_waiting}\}$ for each node $v \in V$, then all the primal and dual variables do not change. Let $\mathbf{x} = \langle x_v \mid v \in V\rangle \in \{0,1\}^n$ and $\mathbf{y} = \langle y_v \mid v \in V\rangle \in \mathbb{R}^n_{\geq 0}$ be the primal and dual solutions derived from the variables $v.x_v$ and $v.y_v$ after convergence, respectively. Let $X = \{v \mid x_v = 1\}$ be the dominating set obtained by Algorithm 3. We divide the set X into two subsets $X_1 = \{v \in X \mid \forall u \in N^+(v) : y_u \leq \lambda w_{\min}(u)\}$, and $X_2 = X - X_1$.

By the construction of Procedure 4, it follows that $\sum_{u\in N^+(v)} y_u \leq w(v)/(1+\varepsilon)$ for each node $v \in X_1$. In addition, each node $v \in X_2$ satisfies $v = \arg\min_{z\in N^+(u)}\{w(z)\}$ for some node $u \in V - N^+(X_1)$. As established in [11], these properties imply that $w(X_1) \leq (2\rho+1)(1+\varepsilon)\sum_{v\in N^+(X_1)} y_v$ and that $w(X_2) \leq (2\rho+1)(1+\varepsilon)\sum_{v\in V-N^+(X_1)} y_v$, thus, $w(X) \leq (2\rho+1)(1+\varepsilon)\sum_{v\in V} y_v$.It follows from Claim 4.13 that $\mathbf{y}$ is feasible. The set X is a $(2\rho+1)(1+\varepsilon)$-approximation for MWDS as a consequence of weak duality. ◀

5 Discussion

In this paper, we presented a new approach for designing self-stabilizing approximation algorithms that is based on the properties of primal and dual LPs. Our approach leaves various open questions for future research. A particularly interesting subject in this context is LP-based algorithms that rely on rounding a fractional solution. Due to their usefulness in the fault-free setting (see, e.g., [13, 25]), we advocate for the study of rounding-based approximation algorithms in the self-stabilizing setting.

References

1 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.

2 Baruch Awerbuch and George Varghese. Distributed program checking: a paradigm for building self-stabilizing distributed protocols. In *FOCS*, volume 91, pages 258–267, 1991.

3 Reuven Bar-Yehuda, Keren Censor-Hillel, Mohsen Ghaffari, and Gregory Schwartzman. Distributed approximation of maximum independent set and maximum matching. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 165–174. ACM, 2017.

4 Reuven Bar-Yehuda, Keren Censor-Hillel, and Gregory Schwartzman. A distributed $(2+\epsilon)$-approximation for vertex cover in o(log Δ / ϵ log log Δ) rounds. *J. ACM*, 64(3):23:1–23:11, 2017.

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

6 Jean RS Blair and Fredrik Manne. Efficient self-stabilizing algorithms for tree networks. In *23rd International Conference on Distributed Computing Systems, 2003. Proceedings.*, pages 20–26. IEEE, 2003.

7 Subhendu Chattopadhyay, Lisa Higham, and Karen Seyffarth. Dynamic and self-stabilizing distributed matching. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 290–297, 2002.

8 Well Y Chiu, Chiuyuan Chen, and Shih-Yu Tsai. A 4n-move self-stabilizing algorithm for the minimal dominating set problem using an unfair distributed daemon. *Information Processing Letters*, 114(10):515–518, 2014.

9 Johanne Cohen, Jonas Lefevre, Khaled Maâmra, Laurence Pilard, and Devan Sohier. A self-stabilizing algorithm for maximal matching in anonymous networks. *Parallel Processing Letters*, 26(04):1650016, 2016.

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

11 Michal Dory, Mohsen Ghaffari, and Saeed Ilchi. Near-optimal distributed dominating set in bounded arboricity graphs. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25–29, 2022*, pages 292–300. ACM, 2022.

12 Swan Dubois and Sébastien Tixeuil. A taxonomy of daemons in self-stabilization. *arXiv preprint*, 2011. `arXiv:1110.0334`.

13 Manuela Fischer. Improved deterministic distributed matching via rounding. *Distributed Computing*, 33, June 2020.

14 Mohsen Ghaffari and Goran Zuzic. Universally-optimal distributed exact min-cut. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25–29, 2022*, pages 281–291. ACM, 2022.

15 Wayne Goddard, Stephen T Hedetniemi, David P Jacobs, Pradip K Srimani, and Zhenyu Xu. Self-stabilizing graph protocols. *Parallel Processing Letters*, 18(01):189–199, 2008.

16 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 *Proceedings International Parallel and Distributed Processing Symposium*, pages 14–pp. IEEE, 2003.

17 Maria Gradinariu and Sébastien Tixeuil. Conflict managers for self-stabilization without fairness assumption. In *27th International Conference on Distributed Computing Systems (ICDCS'07)*, pages 46–46. IEEE, 2007.

18 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.

19 Sandra M Hedetniemi, Stephen T Hedetniemi, David P Jacobs, and Pradip K Srimani. Self-stabilizing algorithms for minimal dominating sets and maximal independent sets. *Computers & Mathematics with Applications*, 46(5-6):805–811, 2003.

20 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.

21 Su-Chu Hsu and Shing-Tsaan Huang. A self-stabilizing algorithm for maximal matching. *Information processing letters*, 43(2):77–81, 1992.

22 Michiyo Ikeda, Sayaka Kamei, and Hirotsugu Kakugawa. A space-optimal self-stabilizing algorithm for the maximal independent set problem. In *the Third International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pages 70–74, 2002.

23 Ken-ichi Kawarabayashi, Seri Khoury, Aaron Schild, and Gregory Schwartzman. Improved distributed approximations for maximum independent set. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 35:1–35:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

24 Jun Kiniwa. Approximation of self-stabilizing vertex cover less than 2. In *Symposium on Self-Stabilizing Systems*, pages 171–182. Springer, 2005.

25 Fabian Kuhn and Rogert Wattenhofer. Constant-time distributed dominating set approximation. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, PODC '03, pages 25–32. Association for Computing Machinery, 2003.

26 Christoph Lenzen, Jukka Suomela, and Roger Wattenhofer. Local algorithms: Self-stabilization on speed. In *Symposium on Self-Stabilizing Systems*, pages 17–34. Springer, 2009.

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

28 Sandeep K Shukla, Daniel J Rosenkrantz, S Sekharipuram Ravi, et al. Observations on self-stabilizing graph algorithms for anonymous networks. In *Proceedings of the second workshop on self-stabilizing systems*, volume 7, page 15. University of Nevada LasVegas, 1995.

29 Gerard Tel. Maximal matching stabilizes in quadratic time. *Information Processing Letters*, 49(6):271–272, 1994.

30 Volker Turau. Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler. *Information Processing Letters*, 103(3):88–93, 2007.

31 Volker Turau and Bernd Hauck. A fault-containing self-stabilizing (3- 2δ+ 1)-approximation algorithm for vertex cover in anonymous networks. *Theoretical computer science*, 412(33):4361–4371, 2011.

32 Vijay V. Vazirani. *Approximation algorithms*. Springer, 2001.

33 Guangyuan Wang, Hua Wang, Xiaohui Tao, and Ji Zhang. A self-stabilizing protocol for minimal weighted dominating sets in arbitrary networks. In *Proceedings of the 2013 IEEE 17th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 496–501. IEEE, 2013.

34 Zhenyu Xu, Stephen T Hedetniemi, Wayne Goddard, and Pradip K Srimani. A synchronous self-stabilizing minimal domination protocol in an arbitrary network graph. In *International Workshop on Distributed Computing*, pages 26–32. Springer, 2003.