社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2005－AL－103 (4)
2005／11／11

# 2より小さい近似比率をもつ自己安定的頂点被覆

木庭 淳
兵庫県立大学経済学部応用経済学科
kiniwa@econ.u-hyogo.ac.jp

**Abstract**

グラフの頂点被覆とは各辺が少なくとも1つの端点をもつような頂点集合である。最小頂点被覆を決定する問題はNP完全であることがよく知られていて近似アルゴリズムが研究されてきた。自己安定アルゴリズムでは現在までに極大マッチングに基づく2-近似解しか知られていなかった。本稿ではネットワークの最大次数 $\Delta$ に対して $(2-1/\Delta)$-近似比をもつ自己安定アルゴリズムを提案する。まず高次数優先極大マッチングを用いた集中型の $(2-1/\Delta)$-近似アルゴリズムを導入し、同じアイデアに基づいた自己安定アルゴリズムを示す。さらに両者が同じ出力結果となることを証明する。

自己安定性、分散頂点被覆、近似アルゴリズム

# Approximation of Self-Stabilizing Vertex Cover Less than 2

Jun Kiniwa
Department of Applied Economics, University of Hyogo
kiniwa@econ.u-hyogo.ac.jp

**Abstract**

A vertex cover of a graph is a subset of vertices such that each edge has at least one endpoint in the subset. Determining the minimum vertex cover is a well-known NP-complete problem in a sequential setting. Thus only a 2-approximation solution based on a self-stabilizing maximal matching has been obviously known until now. In this paper we propose a new self-stabilizing vertex cover algorithm that achieves $(2-1/\Delta)$-approximation ratio, where $\Delta$ is the maximum degree of a given network. We first introduce a sequential $(2-1/\Delta)$-approximation algorithm that uses a maximal matching with the high-degree-first order of vertices. Then we present a self-stabilizing algorithm based on the same idea, and show that the output of the algorithm is the same as that of the sequential one.

self-stabilization, distributed vertex cover, approximation algorithm

## 1 Introduction

Self-stabilization is the most fundamental concept of automatic recovery in distributed systems. A lot of researchers have paid attention to its fault tolerance. A transient fault is the fault that only perturbs the system state, but not the program code. Self-stabilizing algorithms tolerate the transient fault in such a way that the system eventually converges to a legitimate state without any aid of external actions. Thus the algorithms must be designed to run for any initial system state. The execution of self-stabilizing algorithms is guaranteed to repair faulty states and to keep legitimacy thereafter.

Given an undirected graph $G = (V, E)$, a vertex cover is a subset $C \subseteq V$ such that each edge $e \in E$ has at least one endpoint in $C$. It is often required to find the minimum $C$, while it is known to be NP-complete[4]. Instead of finding an exact solution, many approximation algorithms have been developed[7, 13, 15, 17]. Though the problem has been primary considered in sequential algorithms, it also has a wide application to distributed systems, e.g., monitoring link failures[1], placement of agents[11], and managing vector clocks[5]. The placement of such agents or processes cannot be designed in a centralized fashion for large-scale networks. So it is significant to obtain a good approximation distributed algorithm. Furthermore, we would like to add the property of fault-tolerance. Hence our chief concern is to construct a self-stabilizing and as small vertex cover as possible.

Some approximation techniques used in sequential algorithms cannot be applied to distributed algorithms because only local computation is allowed. There, however, seem to be at least two applicable methods — a maximal matching and covering a high-degree vertex first. A maximal matching $M$ is a maximal set of edges with no two of them share the same vertex. It is known that a set of both endpoints of every edge in $M$ form a vertex cover with at most twice the optimum. On the other hand, it is also well-known that the worst example of the greedy method, i.e., covering a high-degree vertex first, is $O(\log n)$ times as large as the optimum. We, however, do not often experience such worst cases when using the greedy method. Instead, it sometimes works better than the maximal matching approximation that always covers both endpoints of the matched edges even if it is not necessary. Hence we combine the greedy method with the maximal matching to ensure the approximation ratio, that is, a maximal matching with the high-degree-first order of vertices.

From the sequential point of view, there have been many approximation approaches, depth-first search[15], a local-ratio theorem[17], semidefinite relaxation[7], and a graph theoretical algorithm[13]. However, Håstad[9] recently showed that there is no $(7/6-\epsilon)$-approximation algorithm for the problem for any $\epsilon > 0$ unless $P = NP$. From the distributed point of view, there does not exist so much work for the vertex cover problem. The significance of this approach, however, has been continuing to grow since large-scale networks, e.g., Internet, mobile ad hoc and sensor networks emerged. Recently, a 2-approximation (weighted) vertex cover algorithm has been proposed[6]. The approximation ratio can be achieved by including matched vertices when a maximal matching is determined. As a part of the vertex cover, algorithms for finding a maximal matching were well-studied. The algorithm presented by Hanckowiack et al.[8] computes a maximal matching in $O(\log^4 n)$ rounds, while the one by Panconesi and Rizzi[14] in $O(\Delta + \log^* n)$ rounds, where $n$ is the number of vertices and $\Delta$ is the maximum degree. A stabilizing maximal matching algorithm was also proposed by Hsu and Huang[12] and its performance was re-evaluated[10, 16], then, the computation model was recently sophisticated[2]. As far as we know, no self-stabilizing vertex cover algorithm with less than 2-approximation ratio is known.

In this paper we develop a self-stabilizing vertex cover algorithm with $(2 - 1/\Delta)$-approximation ratio, where $\Delta$ is the maximum degree of a network. To introduce our idea, we first describe a sequential version of our algorithm and derive its approximation ratio. Next we state our self-stabilizing algorithm and its correctness proofs. Then we claim that our

main algorithm converges to the same solution as the one the sequential algorithm outputs.

The rest of this paper is organized as follows. Section 2 states our self-stabilization model. Section 3 describes a sequential approximation algorithm to prepare for our main algorithm. Section 4 presents our method that finds as small a vertex cover as possible. Section 5 proves the correctness of our algorithm. Finally, Section 6 concludes the paper.

## 2 Model

Here we describe a model used in our discussion. There are $n$ processes $P = \{1, 2, \ldots, n\}$, where each process is identified by the hard-wired unique name. A network consists of $n$ processes of finite state machines connected arbitrarily with bidirectional communication links, where each process $i \in P$ has shared state variables with a finite set of states $\Sigma_i$. We assume that transient faults sometimes occur at the variables and the interval between faults is sufficiently long compared to a stabilization time. The global state of all processes is called a *configuration*. The set of all configurations is denoted by $\Gamma = \Sigma_1 \times \Sigma_2 \times \cdots \times \Sigma_n$.

A network is represented by a graph $G = (V, E)$, where $V(= P)$ is a set of vertices (processes) and $E$ is a set of edges (communication links). For convenience, we use the corresponding terms process–link and vertex–edge interchangeably. A vertex $i \in V$ is *adjacent* to another vertex $j$ and an edge $e \in E$ is *incident* to $i$ or $j$ if there is an edge $e = (i, j)$. Each process $i$ is assumed to maintain a set of adjacent processes $N(i)$ correctly. Let $N^+(i) = \{i\} \cup N(i)$ denote process $i$ with its adjacent processes. The number of edges incident to vertex $i$ is called a *degree* (or particularly called a *pure degree*), denoted by $\delta_i \leq \Delta$, where $\Delta$ is the maximum (pure) degree of a graph $G$. We define an *identified degree* $d_i$ as $(\delta_i, i)$ to make the degree of a vertex unique. Then we say that $d_k < d_i$, called $d_k$ is *lower* than $d_i$ (or $d_i$ is *higher* than $d_k$), if $\delta_k < \delta_i$ or $(\delta_k = \delta_i) \wedge (k < i)$. A vertex $i$ with the *highest* (*maximum*) degree means that there is no vertex with higher degree in $N(i)$ for distributed algorithms, or in $V \backslash i$ for sequential algorithms. The *lowest* (*minimum*) degree is similarly defined. When we just refer to a degree without notice, e.g., high-degree-first, it means the identified degree[1].

We assume a *state-reading model* for simplicity, that is, each process directly reads the shared variables of adjacent processes and updates only the variables of its own. Each process has a program of internal computations, "**if** *Guard* **then** *Action*", or denoted by *Guard* $\Rightarrow$ *Action*. If *Guard* is true in a process $i$, the process is said to be *enabled*. A transient fault may perturb states of processes, where the number of enabled processes may be more than one. From the set of enabled processes $EP \subseteq P$, a scheduler *D-daemon* (distributed daemon) selects, or called *activates*, a non-empty set of processes $A \subseteq EP$ at a configuration $c_j \in \Gamma$. An *atomic step* consists of reading the states of adjacent processes, an internal computation, and writing its own state. We say that $c_{j+1}$ is reached from $c_j$ for such a transition of configurations. An *execution* $E$ is a sequence of configurations $E = c_0, c_1, \ldots, c_j, c_{j+1}, \ldots$ such that $c_{j+1} \in \Gamma$ is reached from $c_j \in \Gamma$. The definition of *round* complexity[3] is as follows. The first round $e$ is the minimal prefix of an execution $E$ in which every process executes at least one action. Let $e'$ be the suffix of $E$ such that $E = ee'$. The second round $e''$ of $E$ is the first round of $e'$, and so forth. The daemon is assumed to be *fair*, that is, every process is activated infinitely often.

The state variables of each process contain a pointer. The pointer *points to* one of adjacent processes to make a matching. We say that process $i$ makes a *proposal* to $k \in N(i)$ if $i$ points to $k$, denoted by $i \rightarrow k$, when $k$ does not point to $i$. Conversely, we say that process $i$ *accepts* a proposal of $k \in N(i)$ if $i$ points to $k$ against the $k$'s proposal to $i$. That is, $i \rightarrow k$ and $k \rightarrow i$ hold and are abbreviated to $i \leftrightarrow k$. In that case, we say that process $i$ is *matched* (with process $k$) and it is denoted by $i \in Matched$ or $(i, k) \in Matched$. If $i \notin Matched$, it is called *unmatched*. A set of matched processes are partitioned into two subsets, *Proposer* and *Acceptor*,

[1] We explicitly use the pure degree only in Lemma 1.

where processes in *Proposer* have made proposals and those in *Acceptor* have accepted them. A vertex is said to be *covered* when it is contained in the vertex cover $C \subseteq V$, or otherwise *uncovered*. An edge is said to be covered when it is incident to some vertex in $C$.

## 3 Preliminaries

To introduce our self-stabilizing algorithm, here we state an underlying sequential algorithm. Our basic idea is partly based on a simple greedy approach : (1) take a vertex $v$ with the maximum degree and delete $v$ with its all incident edges, and (2) iterate this until there are no edges left. Though the greedy method seems to be usually good, its approximation ratio is known to be $O(\log n)$ times the optimal vertex cover in the worst case.

On the other hand, our idea is also partly based on a 2-approximation algorithm that makes use of a maximal matching. This approximation ratio is derived by the fact that a vertex cover is always included in any maximal matching. Since the algorithm always outputs both the matched vertices, the approximation is not good if the matching is close to the maximum matching. The bounded approximation ratio, however, is attractive to researchers.

Hence our method is to combine these algorithms so that it usually works well with the guarantee of the bounded worst case. Furthermore, we would like to achieve a better approximation ratio than 2, which is known to be the best one in a distributed setting[6]. First, we construct a high-degree-first maximal matching, where the vertex with the maximum degree is matched first, and then cover the vertex (one endpoint of the matched edge), and so forth. Finally, we cover some vertices in order to complete a vertex cover by using the information of degrees.

Now we outline a sequential algorithm **VCover**, in which we use some expressions similar to the self-stabilizing algorithm. Let $L$ be a sorted list whose top is the vertex with the maximum degree. A vertex $v$, the top element of $L$, is iteratively selected. Since the selected vertex $v$ corresponds to the process making a proposal, we call $v$ a *proposal vertex* and express $v \in$ *Proposer*. Likewise, the minimum degree vertex $u$ matched with the proposal vertex $v$ is called an *acceptance vertex*. We call the $i$-th selected vertex in *Proposer* the *$i$-th proposal vertex*.

1. Sort vertices into the high-degree-first order and construct a list $L$ according to the order.
2. For each vertex $v$ at the top of $L$, iterate (a)–(c) (until no vertex can be selected):
   (a) Select its adjacent vertex $u$ with the minimum degree and join $(u, v)$ to a matching.
   (b) Cover the vertex $v$ (not $u$).
   (c) Remove $u$ and $v$ from the list $L$.
3. If there is some uncovered vertex $v \notin$ *Proposer* which is adjacent to some uncovered vertex $u$ with a lower degree, cover the vertex $v$.

The following lemma shows the performance of **VCover**. Let $Inc(i)$ be a set of incident edges to vertex $i$. The set $Inc(i)$ is partitioned into three disjoint subsets, i.e., $Inc(i) = Old(i) \cup New(i) \cup Both(i)$, where $|Old(i)| = m_i$ and $|New(i) \cup Both(i)| = k_i$. Hence $\delta_i = m_i + k_i$. For simplicity, we denote $New(i) \cup Both(i)$ by $E(i)$. The subsets are defined as

$$Old(i) = \{(i,j) \mid \delta_i \leq k_j\},$$
$$New(i) = \{(i,j) \mid \delta_j \leq k_i\} \text{ and}$$
$$Both(i) = \{(i,j) \mid k_j < \delta_i,\ k_i < \delta_j\}.$$

In other words, $(i,j) \in Old(i)$ is equivalent to $(i,j) \in New(j)$ because $k_i < \delta_i \leq k_j \leq \delta_j$ holds. Conversely, $(i,j) \in Old(j)$ is equivalent to $(i,j) \in New(i)$. To estimate the performance of **VCover**, an identical cost defined below is given for $(i,j) \in Old(i) \cap New(j)$ and $(i,j) \in New(i) \cap Old(j)$. On the other hand, $(i,j) \in Both(i)$ is equivalent to $(i,j) \in Both(j)$, for which separate costs are given.

**Lemma 1.** *The approximation ratio of the algorithm* **VCover** *is*

$$2 - \frac{1}{\Delta}.$$

*Proof.* The proof proceeds by assigning costs $c_{ij}$ and $c_{ji}$ to each edge $(i,j) \in E(i) = New(i) \cup Both(i)$, and then using the costs to derive the relationship between the size of the optimal vertex cover $|C^*|$ and the size of the output $|C|$ of **VCover**. The cost $c_{ij}$ is defined as evenly distributing the cost of 1 over every edge $(i,j) \in E(i)$. Notice that any edge $(i,j) \in Both(i) \cap Both(j)$ has two separate costs $c_{ij} = 1/k_i$ and $c_{ji} = 1/k_j$, while the other edge $(i,j) \in New(i) \cap Old(j)$ has the identical cost $c_{ij} = c_{ji} = 1/k_i$.

We can construct the three subsets of edges in the process of **VCover**. In most cases, the edges covered for the first time by $i$ belong to $E(i)$ because the high-degree-first ordered list $L$ is used. The only exception is that the edge $(i,j)$ firstly covered by $i$ may belong to $New(j)$. Since this case requires both $i$ and $j$ to be covered due to $\delta_i \leq k_j$, shifting the "firstly covered vertex" to $j$ unlike **VCover**, does not change the number of covered vertices. Hence the sum of the entire edge costs $\sum_{(i,j)\in E(i)} c_{ij}$ for all vertex $i \in V$ is equivalent to $|C|$.

The sum of costs for $k_i$ edges in $E(i)$ is $k_i \cdot 1/k_i = 1$ (see Fig. 1). On the other hand, the edge cost $c_{ji} = 1/k_j$ for an edge $(i,j) \in Old(i)$ is bounded by $1/\delta_i$. Since there are $m_i$ edges in $Old(i)$, the entire cost for $Inc(i)$ is

$$\sum_{(i,j)\in Inc(i)} c_{ij} \leq 1 + \frac{m_i}{\delta_i} = 2 - \frac{k_i}{\delta_i}.$$

Thus we obtain

$$\begin{aligned} |C| &\leq \sum_{i\in C^*} \sum_{(i,j)\in Inc(i)} c_{ij} \\ &\leq |C^*|(2 - \frac{1}{\Delta}). \end{aligned}$$

□

## 4 Self-Stabilizing Algorithm

Now we present our self-stabilizing algorithm **SSVC**. The **SSVC** is a distributed version of the **VCover** stated above. To execute operations locally, some techniques are contained in the algorithm.

First, each process has a variable *color*, denoted by `col`, in order to construct the high-degree-first matching. The color is defined as the identified degree of a process if not matched, and as the same color as that of the proposal process if matched. If an unmatched process can detect some adjacent, lower colored processes, it makes a proposal to the minimum degree one. Thus the color determines whether or not it can make a proposal. Even if more than one processes concurrently make proposals to the same process $i$, the process $i$ accepts the proposal of the maximum degree process $j$. Then process $i$'s color is boosted up to process $j$'s one. Thus lower colored processes cannot make proposals to $i$ thereafter. After $i$ has been matched with $j$, the unmatched, proposal processes must give up their proposals to $i$. To make it possible, we use a totally ordered degree, i.e., an identified degree, for each process.

Second, every process that can make a proposal is covered with respect to the vertex covering. Then the remaining covered processes are determined as follows. Let $i$ be a non-proposal process. If every adjacent process $k \in N(i)$ has been matched with $j \neq i$ and has higher degree than $i$, process $i$ is not covered. Otherwise, $i$ is covered.

We use a shared variable $\texttt{col}_i \in \{(\delta_j, j) \mid 1 \leq \delta_j \leq \Delta,\ j \in P\}$ representing process $i$'s color, and $\texttt{col}_i = (\delta_i, i)$ when $i$ is unmatched. Each process $i$ has a variable $\texttt{cover}_i \in \{true, false\}$, representing a covered process when $\texttt{cover}_i = true$, which may not be shared.

The **SSVC** is formally described as follows.

**definition of sets**

$$\begin{aligned} Low_i &= \{k \in N(i) \mid \texttt{col}_k < \texttt{col}_i\} \\ High_i &= \{k \in N(i) \mid \texttt{col}_i < \texttt{col}_k\} \\ Other_i &= \{k \in N(i) \mid k \to j \neq i\} \\ dmin(i) &= \{k \mid \min_{k\in Low_i} d_k\} \\ dmax(i) &= \{k \mid \max_{k\in High_i} d_k,\ k \to i\} \end{aligned}$$

Notice that $dmin(i)$ is the minimum degree vertex among the lower colored vertices adjacent to $i$, and that $dmax(i)$ is the maximum degree vertex among the higher colored adjacent vertices that point to $i$.

**high-degree-first matching**

$$(\forall k \in N(i) : i \not\leftrightarrow k) \wedge (\mathtt{col}_i \neq d_i) \Rightarrow \mathtt{col}_i := d_i \quad \text{(a)}$$

$$(\exists k \in N(i) : i \leftrightarrow k) \wedge (\mathtt{col}_i \neq \max(d_i, d_k)) \Rightarrow \mathtt{col}_i := \max(d_i, d_k) \quad \text{(b)}$$

$$\exists dmax(i) : (dmax(i) \rightarrow i) \wedge (i \not\rightarrow dmax(i)) \Rightarrow i \rightarrow dmax(i)\ ;\ \mathtt{col}_i := d_{dmax(i)}\ ;\ \mathtt{cover}_i := false \quad \text{(c)}$$

$$\exists k \in High_i : (i \rightarrow k) \wedge (k \not\rightarrow i) \Rightarrow i \rightarrow null\ ;\ \mathtt{col}_i := d_i\ ;\ \mathtt{cover}_i := false \quad \text{(d)}$$

$$(\exists dmax(i) : (dmax(i) \not\rightarrow i)) \wedge (\exists dmin(i) : (i \not\rightarrow dmin(i))) \Rightarrow i \rightarrow dmin(i)\ ;\ \mathtt{cover}_i := true \quad \text{(e)}$$

**non-proposal covered vertices**

$$(\forall k \in Other_i : (d_i < d_k) \wedge \mathtt{cover}_i) \vee (\exists k \in Other_i : (d_k < d_i) \wedge (\neg\mathtt{cover}_i)) \Rightarrow \mathtt{cover}_i := \neg\mathtt{cover}_i \quad \text{(f)}$$

Each statement above is informally explained in what follows.

(a) Every wrong color of unmatched processes is corrected.
(b) Every wrong color of matched processes is corrected.
(c) The proposal of a process with the (locally) maximum degree is accepted if at least one proposal is made.
(d) A proposal to a higher colored process is discarded.
(e) If a process is not pointed by any higher degree process and is adjacent to some lower colored processes, it makes a proposal to the minimum degree process among them.
(f) If a process $i$ is adjacent to only processes with pointing others and $i$ has the minimum degree among them, it is not covered. Conversely, if such a process does not have the minimum degree, it is covered.

## 5 Correctness and Properties

To show that **SSVC** is deadlock-free, we introduce the following predicate. When $\mathcal{VC}$ is true, it means the system reaches a legitimate configuration.

$$\mathcal{VC} \equiv (\forall e = (i,j) \in E : (\mathtt{cover}_i = true) \vee (\mathtt{cover}_j = true))$$

**Lemma 2.** *If $\mathcal{VC}$ is false, there exists at least one process which can apply some rule in* **SSVC**.

*Proof.* **(by contradiction)** Suppose that there is some edge $e = (i,j)$ whose both ends are not covered. Then neither of them has executed (e). There are two cases.

(1) Neither $i$ nor $j$ is in *Matched* :
Since $d_i < d_j$ or $d_j < d_i$ holds, the higher degree process between $i$ and $j$ can execute (e).
(2) Either $i$ or $j$ is in *Matched* :
Suppose that $i$ is matched and $j$ is not. If $d_i < d_j$, then $(\exists i \in Other_j : (d_i < d_j) \wedge (\neg\mathtt{cover}_j))$ in (f) is true for process $j$. Otherwise, the same rule is applied to process $i$.

Therefore, every edge is eventually covered. □

To show that **SSVC** converges to a legitimate configuration, we define the following pseudo-legitimate states and use the proof method of *convergence stairs*.

$$Eligible = \{i \mid Low_i \neq \phi\},$$
$$Lowest = \{i \mid \forall k \in Other_i : d_i < d_k\}$$

$$P1 \equiv (\forall i \notin Matched : (\mathtt{col}_i = d_i)) \wedge (\forall (i,k) \in Matched : (\mathtt{col}_i = \max(d_i, d_k)))$$

$$P2 \equiv P1 \wedge (\forall i \in Eligible : (i \leftrightarrow dmin(i)) \wedge (\mathtt{cover}_i = true))$$

$$P3 \equiv P2 \wedge (\forall i \in \neg Proposer : (\neg Lowest : (\mathtt{cover}_i = true) \wedge Lowest : (\mathtt{cover}_i = false)))$$

**Lemma 3.** *P1 eventually holds.*

*Proof.* Let $i$ be a process in $\neg Matched$. If $i$ has a wrong $\mathtt{col}_i$, that is, $\mathtt{col}_i \neq d_i$, then it is corrected by (a). Suppose that $i$ moves from $\neg Matched$ to $Matched$. If several processes in $High_i$ point to $i$, then $i$ selects the process $dmax(i)$ with the maximum degree among them and sets $\mathtt{col}_i$ to the same degree as $dmax(i)$ by (c). Conversely, if $i$ is pointed by a process $k \in Low_i$, $i$ just points to $k$ but $\mathtt{col}_i$ remains unchanged. Thus $\forall (i,k) \in Matched : (\mathtt{col}_i = \max(d_i, d_k))$ holds.

On the other hand, let $i$ be a matched process with $k$. If $i$ has a wrong $\mathtt{col}_i$, that is, $\mathtt{col}_i \neq \max(d_k, d_i)$, then it is corrected by (b). Suppose that $(i,k)$ has changed from $Matched$ to $\neg Matched$. If $k$ changes its pointer to another process, then $\mathtt{col}_i$ is reset to $d_i$ (if necessary) by (d). Thus $\forall i \notin Matched : (\mathtt{col}_i = d_i)$ holds.

After every process has been activated, P1 is satisfied. □

**Lemma 4.** *P2 eventually holds.*

*Proof.* **(by induction)** Suppose that $P1 \wedge \neg P2$ holds. Hence every information about col is true. Let $K = \{i_1, i_2, \ldots, i_k\}$, where $d_{i_1} > d_{i_2} > \cdots > d_{i_k}$, be a set of processes such that $i \in K$ belongs to $Eligible$ and $k \in N^+(i) : k \not\leftrightarrow dmin(k)$. Let $j$ represent $dmin(i_1)$ for simplicity. Without loss of generality, we assume that $i_2 \leftrightarrow j\ (= dmin(i_1))$ and $d_{i_2} > d_j$. Since $i_1 \not\leftrightarrow dmin(i_1)$, $i_1$ eventually points to $j$, that is, $i_1 \rightarrow j$, and $\mathtt{cover}_{i_1} := true$ by rule (e). Since $\mathtt{col}_j = \mathtt{col}_{i_2} < \mathtt{col}_{i_1}$, rule (c) can be applied to $j$. Thus $i_1 \leftrightarrow j$ holds and $i_1$ is removed from $K$. At this time, $\mathtt{col}_j := d_{i_1}$ and thus $\mathtt{col}_j > \mathtt{col}_{i_2}$ holds. Then by rule (d), $i_2 \rightarrow null$ holds. If $Low_{i_2}$ is not empty, $i_2$ can select $dmin(i_2)$ from $Low_{i_2}$.

Notice that if $Low_i$ becomes empty for some $i \in K$, such $i$ is automatically removed from $K$. Since the same argument holds for any $i_j, i_{j+1} \in K$ as long as they belong to $Eligible$, the lemma follows by induction. □

**Lemma 5.** *P3 eventually holds.*

*Proof.* Suppose that $P2 \wedge \neg P3$ holds and that some process $i$ in $\neg Proposer \wedge \neg Lowest$ has $\mathtt{cover}_i = false$. Then the portion $(\exists k \in Other_i : (d_k < d_i) \wedge (\neg \mathtt{cover}_i))$ in rule (f) will correct such process $i$. Next suppose that some process $i$ in $\neg Proposer \wedge Lowest$ has $\mathtt{cover}_i = true$. Then the portion $(\forall k \in Other_i : (d_i < d_k) \wedge \mathtt{cover}_i)$ in rule (f) corrects such an error. □

Let $M_s$ be a matching eventually determined by **SSVC**. The following theorem shows the efficiency of our method.

**Theorem 1. SSVC** *is a self-stabilizing distributed vertex cover algorithm whose stabilization time is* $|M_s| + 2$ *rounds.*

*Proof.* For the $j$-th round, the $j$-th highest-degree vertex makes a correct proposal and the $(j-1)$-st proposal is accepted. Hence, a maximal matching is completed in the $(|M_s|+1)$-st round. Since non-proposal vertices, which have some adjacent lower degree vertices, locally know the necessity of being covered, they are covered in the $(|M_s|+2)$-nd round. □

We show Lemmas 6 and 7 which claim the output of our self-stabilizing algorithm is equivalent to that of the sequential one.

**Lemma 6.** *Let $M_v$ be a matching determined by* **VCover**. *We claim*

$$M_s = M_v.$$

*Proof.* **(by induction)** Let $i_k$ be the $k$-th proposal vertex, and $i_s^k \in N(i_k)$ the matched vertex with $i_k$ in **VCover**. Since $i_s^1$ is the vertex with the minimum degree in $N(i_1)$, the edge $(i_1, i_s^1)$ belongs to $M_v$. On the other hand, $i_1$ makes a proposal to $i_s^1$ by rule (e) in **SSVC**. Then the proposal is eventually accepted irrespective of whether $i_s^1$ has already matched or not. Hence the edge $(i_1, i_s^1)$ also belongs to $M_s$.

Suppose that every vertex $\{i_1, \ldots, i_k\}$ in **SSVC** has the same edge $\{i_s^1, \ldots, i_s^k\}$ as in **VCover**. Further suppose that $(i_{k+1}, i_s^{k+1})$ belongs to $M_v$. Then $i_s^{k+1}$ is the vertex with the minimum degree in $N(i_{k+1})$. In **SSVC**, if $i_{k+1}$ has made a proposal to $i_s^t \in \{i_s^1, \ldots, i_s^k\}$, $i_s^t$ will reject the proposal and then $i_{k+1}$ will eventually select the vertex with the minimum degree by rules (d) and (e). Since the vertex with the

minimum degree in $N(i_{k+1})$ is uniquely determined, the edge $(i_{k+1}, i_s^{k+1})$ will also be contained in $M_s$. □

**Lemma 7.** *Let $C_s$ and $C_v$ be sets of vertex cover determined by* **SSVC** *and* **VCover**, *respectively. We claim*

$$C_s = C_v.$$

*Proof.* By Lemma 6, $C_s = C_v$ holds with respect to *Proposer* in $M_s$ and $M_v$. The remaining covered vertices are uniquely determined by only degree information, which is equivalent in two algorithms. Thus $C_s = C_v$ also holds in the remaining portion. □

By Lemmas 1 and 7, we obtain the following theorem.

**Theorem 2.** *The approximation ratio of* **SSVC** *is* $2 - 1/\Delta$. □

## 6 Conclusion

We proposed a self-stabilizing algorithm for finding as small a vertex cover as possible in distributed systems. It has a wide application to the placement of agents or facilities in networks. The obtained approximation ratio is at most $2 - 1/\Delta$. It is interesting that the priority of vertices generates the same output as a sequential algorithm.

## Acknowledgement

The author would like to thank Prof. Kensaku Kikuta for useful discussions and helpful comments. This work was partially supported by Grant-in-Aid for Scientific Research ((C)17510131) of the Ministry of Education, Science, Sports, and Culture of Japan.

## References

[1] Y.Bejerano and R.Rastogi, Robust monitoring of link delays and faults in IP networks, In *Proceedings of the IEEE INFOCOM*, March (2003).

[2] S.Chattopadhyay, L.Higham and K.Seyffarth, Dynamic and self-stabilizing distributed matching, In *Proceedings of the 21st Annual ACM Symposium on Principles of Distributed Computing*, July (2002) 290–297.

[3] S.Dolev, A.Israeli and S.Moran, Uniform dynamic self-stabilizing leader election, *IEEE Transactions on Parallel and Distributed Systems*, **8**(4) (1997) 424–440.

[4] M.R.Garey and D.S.Johnson, Computers and intractability. a guide to the theory of NP-completeness, *Freemann* (1979).

[5] V.K.Garg and C.Skawratananond, On timestamping synchronous computations, In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS'02)* , July (2002).

[6] F.Grandoni, J.Könemann, and A.Panconesi, Distributed weighted vertex cover via maximal matchings, In *Proceedings of the 11th International Computing and Combinatorics Conference (COCOON'05)*, August (2005).

[7] E.Halperin, Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs, *SIAM Journal on Computing* **31**(5) (2002) 1608–1625.

[8] M.Hańćkowiak, M.Karoński, and A.Panconesi, On the distributed complexity of computing maximal matchings, *SIAM Journal on Discrete Mathematics* **15**(1) (2001) 41–57.

[9] J.Håstad, Some optimal inapproximability results, *Journal of the ACM*, **48**(4), (2001) 798–859.

[10] S.T.Hedetniemi, D.P.Jacobs, and P.K.Srimani, Maximal matching stabilizes in time $O(m)$, *Information Processing Letters* **80** (2001) 221–223.

[11] T.A.Hegazy, A distributed approach to dynamic autonomous agent placement for tracking moving targets with application to monitoring urban environments, Ph.D. Thesis, *School of Electrical and Computer Engineering, Georgia Institute of Technology* (2004).

[12] S.-C.Hsu and S.-T.Huang, A self-stabilizing algorithm for maximal matching, *Information Processing Letters* **43** (1992) 77–81.

[13] H.Nagamochi and T.Ibaraki, An approximation of the minimum vertex cover in a graph, *Japan Journal of Industrial and Applied Mathematics* **16** (1999) 369–375.

[14] A.Panconesi and R.Rizzi, Some simple distributed algorithms for sparse networks, *Distributed Computing*, **14**, (2001) 97–100.

[15] C.Savage, Depth-first search and the vertex cover problem, *Information Processing Letters* **14**(5) (1982) 233–235.

[16] G.Tel, Maximal matching stabilizes in quadratic time, *Information Processing Letters* **49** (1994) 271–272.

[17] R.Bar-Yehuda and S.Even, A local-ratio theorem for approximating the weighted vertex cover problem, *Annals of Discrete Mathematics*, **25** (1985) 27–46.