Vol.2010-MPS-77 No.10
2010/3/4

A self-stabilizing distributed algorithm for the multicoloring problem in dynamic networks

Hirotsugu Kakugawa [†1] and Yukiko Yamauchi [†2]

The problem of channel (frequency) assignment is an important problem in wireless networks such as cellular networks and wireless LANs. In this paper, we consider a problem of multiple channel assignment to each node in a wireless network, and we formalize this problem as the multicoloring problem of a graph; each node is assigned a set of colors and no two neighbor nodes share the same color. We propose a self-stabilizing distributed algorithm for the multicoloring problem. By the self-stabilizing property, our algorithm is adaptive to topology changes, it tolerates any kind and any finite number of transient faults, and it does not require global reset to recompute multicoloring. Additionally, our algorithm has several desirable properties for dynamic distributed systems.

1. Introduction

The problem of channel (frequency) assignment is an important problem in wireless networks such as cellular networks and wireless LANs. In this paper, we consider a problem of multiple channel assignment to each node or access point in a wireless network. We formalize this problem as the *multicoloring problem* of a graph in such a way that a node is an access point, an edge is an interference relation between a pair of access points, and a set of colors assigned to each node corresponds to a set of channels (frequencies) assigned to each access point. This problem is a generalization of the coloring problem that is a formulation of the (single) channel assignment problem.

Autonomous distributed channel assignment is an crucial issue in wireless networks, especially, wireless LANs and wireless mesh networks. Recently, such wireless networks are getting popular, and many access points (APs) are deployed densely without coordination. Although channel assignment can be optimized and managed within an organization, neighboring organizations are not always cooperative, and as a result, a channel conflict problem arises. A centralized algorithm for channel assignment requires a computing resource for the computation of combinatorial optimization and it is difficult to make each organization to cooperate with neighboring organizations. Hence, a desired approach to solve this problem is to let each AP select channels autonomously in a distributed manner. In this paper, we propose an autonomous distributed algorithm for this problem based on a theoretical framework called self-stabilization.

The concept of *self-stabilization* is proposed by Dijkstra[1] as one of theoretical basises of autonomous fault-tolerant distributed algorithm. Starting from arbitrarily initial system state (global state of a distributed system), a self-stabilizing distributed algorithm autonomously recovers to correct system state. Here, “arbitrary” initial system state means that the initial state of each node can be completely at random. Hence, a self-stabilizing distributed algorithm has the following three important properties.

- It tolerates any kind and any number of transient faults. By transient fault, we mean a soft error such as message loss, message corruption, and memory corruption. The system state becomes arbitrary by transient faults, however, self-stabilization guarantees recovery from such arbitrary system state.
- It is adaptive to changes of network topology such as deployment and removal of nodes, and connection and disconnection of communication links. Although such events lead a distributed system to an incorrect system state, self-stabilization guarantees that the system state reaches a correct one for new topology.
- It does not require synchronized initialization of nodes. Although a non-self-stabilizing distributed algorithm requires that all the nodes must be initialized in a synchronized manner when it is started, a self-stabilizing algorithm does not require any initialization; this is because it can start from arbitrary initial system state.

1.1 Related works

Since the problem of assignment of single channel for each access point is formalized by the problem of node coloring of a graph, there are many literature in

†1 Graduate School of Information Science and Technology, Osaka University, Japan
†2 Graduate School of Information Science, Nara Institute of Science and Technology, Japan

various problem settings.

In 2)–5), self-stabilizing algorithms for the (single-)coloring problem are proposed. To the best knowledge of authors, there is no literature that proposes a self-stabilizing distributed algorithm for the multicoloring problem. The algorithm proposed in 2) works on planer graphs and uses with six colors. The algorithm proposed in 3) works on bipartite graphs. The algorithm proposed in 4) works in arbitrary graph topology, uses $\Delta+1$ colors, where Δ is the maximum degree of a node in the graph, and convergence time is $O(n\Delta)$, where n is the number of nodes. The algorithm proposed in 5) works in arbitrary graph topology, uses $\Delta+1$ colors, and convergence time is linear to n.

Although the original definition of self-stabilization guarantees convergence (i.e., recovery) to a correct system state from arbitrary initial system state caused by any scale of faults, it does not guarantee any property (1) until convergence is finished, and (2) even if small topology change occurs when the system state is correct, for example. To enhance the property of self-stabilization in such dynamic networks, following extensions are proposed in the literature: superstabilization[6], output stability[7),8], and safe convergence[9].

1.2 Contribution of this paper

In this paper, we propose a self-stabilizing distributed algorithm for the multicoloring problem with superstabilization, output stability, and safe convergence properties. To the best of our knowledge, this is the first one. Our algorithm adopts a high-level computational model and many technical details required for implementation are not considered. However, our algorithm gives a theoretical basis of autonomy of distributed algorithms for multichannel assignment.

1.3 Organization of this paper

In section 2, definition of self-stabilization and problem statement are given. In section 3, we propose a self-stabilizing multicoloring algorithm. Proofs are omitted due to space limitation, and they will be found in the full version. In section 4, we discuss the number of available colors for self-stabilizing algorithms based on local-greedy strategy. Then, we discuss the behavior of the proposed algorithm when the number of colors is small. In section 5, we show that multicoloring property is maintained by dynamic change of network, and we show superstabilization, output stability, and safe convergence of the proposed algorithm. In section 6, we give concluding remarks.

2. Definitions

Let V be a set of processes and E be a set of bidirectional links of a distributed system. Then, a network topology of a distributed system is represented by a graph $G(V,E)$. We assume that each process $P_i \in V$ is given a set of neighboring process $N_i = \{P_j \in V : (P_i, P_j) \in E\}$, and each P_i can communicate each $P_j \in N_i$. Let the degree of P_i denoted by δ_i be $|N_i|$, and the maximum degree denoted by Δ be $\max_{P_i \in V} \delta_i$.

2.1 Self-stabilization

Each process P_i maintains a set of local variables. Let q_i be a local state (a tuple of all local variables) of process P_i. A tuple $(q_1, q_2, \dots, q_n)$ of local states of all the processes, where $n = |V|$, is a *configuration* (global state) of a distributed system. Let Γ be the set of all configurations.

An algorithm at each process P_i is given as a finite set of guarded commands in the following form:

$$Grd_1 \to Act_1;\ Grd_2 \to Act_2; \quad \cdots \quad Grd_L \to Act_L;$$

Each Grd_ℓ ($\ell = 1, 2, ..., L$) is called a *guard* and it is a predicate on P_i's local state and local states of its neighboring processes. For communication model, we assume that each process can read local states of neighbors, which is called the *state-reading model.* Although a process can read local states of neighbors, it can update its local state only. We say that P_i is *enabled* in configuration $\gamma \in \Gamma$ if and only if at least one guard of P_i is true in γ. If P_i is not enabled, we say that P_i is *disabled.* Each Act_k is called *action* or *move* which updates the local state of P_i, and the next local state is computed from the current local state of P_i and those of its neighboring processes.

For local computation, we assume the *composite atomicity* model: a scheduler selects an enabled process and let it execute one of the corresponding action in an atomic step. This type of execution scheduler is known as *the central daemon.*

For any configuration γ, let γ' be a configuration that follows γ by selecting a process. Then, we denote by $\gamma \to \gamma'$ the transition relation between configurations. We denote by $\gamma \xrightarrow{*} \gamma'$ if and only if $\gamma_0 (= \gamma) \to \gamma_1$, $\gamma_1 \to \gamma_2$, $\dots$, $\gamma_{l-1} \to \gamma_l (= \gamma')$ for some $l \geq 0$. We say γ' is *reachable* from γ if and only if

$\gamma \xrightarrow{*} \gamma'$ holds.

A *computation* starting from configuration γ_0 is a (possibly infinite) sequence of configuration $E = \gamma_0, \gamma_1, \gamma_2, \ldots$ such that γ_{t+1} is the next configuration of γ_t, i.e., $\gamma_t \rightarrow \gamma_{t+1}$ for each $t \geq 0$. A computation terminates if there is no enabled process. Otherwise, a computation is infinite. Note that there may be more than one process to be selected in each configuration because there may be more than one enabled processes. Hence, huge number of computations are possible because of the choice of process by the central daemon.

Definition 1 (Self-stabilization[1]) An algorithm is *self-stabilizing* with respect to $\Lambda \subseteq \Gamma$ if and only if the following two conditions hold.

- Convergence. Starting from any initial configuration $\gamma \in \Gamma$, computation eventually reaches a configuration in Λ.
- Closure. Starting from any initial configuration $\gamma \in \Lambda$, any configuration reachable from γ is in Λ.

Each $\gamma \in \Lambda$ is called a *legitimate* configuration. □

We assume that the central daemon is *unfair*, i.e., there is no fairness in selection of processes for execution, and the central daemon selects an enabled process for execution arbitrarily in order to prevent convergence. Hence, an algorithm must guarantee convergence for any (malicious) scheduling.

2.2 Problem statement

The number of request of colors at each process is represented by a function w. Let $w^{\max} = \max_{P_i \in V} w(P_i)$. Let C be a set of colors, and we assume that $|C| \geq (\Delta + 1)w^{\max}$ holds. (Later, we consider a case $|C| < (\Delta + 1)w^{\max}$.)

Definition 2 (Multicoloring[10]) For a network $G(V, E)$ and a set of colors C, a function f, which assigns to each $P_i \in V$ a subset of colors $f(P_i) \subseteq C$, is a *multicoloring* if and only if $\forall (P_i, P_j) \in E : f(P_i) \cap f(P_j) = \emptyset$. □

Definition 3 (Proper multicoloring[10]) For a vertex weighted network $G(V, E, w)$ and a set of colors C, a function f, which assigns to each $P_i \in V$ a subset $f(P_i) \subseteq C$, is a *proper* multicoloring if and only if

- f is multicoloring, and
- $\forall P_i \in V \ : \ |f(P_i)| = w(P_i)$. □

Definition 4 (The distributed proper multicoloring problem) For a vertex weighted network $G(V, E, w)$ and a set of colors C, the *distributed proper multicoloring problem* is a problem such that

- Each $P_i \in V$ maintains a local variable $f_i (\subseteq C)$,
- Each $P_i \in V$ computes a multicolor assignment in f_i, and
- Collection of f_i is a proper multicoloring, that is, the function f, where $f(P_i) = f_i$ for each $P_i \in V$, is a proper multicoloring for $G(V, E, w)$ and C. □

2.3 Extensions of self-stabilization for dynamic networks

Self-stabilization does not guarantee safety and stability of a system during convergence after transient faults and topology changes. Superstabilization[6], output stability[7],[8], and safe convergence[9] are extensions of self-stabilization to enhance safety and stability in dynamic networks.

2.3.1 Superstabilization

Superstabilization[6] is one of the extensions of self-stabilization for dynamic networks. It guarantees some property of the output of a system immediately after a fault or a topology change that occurs in a legitimate configuration. In 6), the concept of superstabilization is explained as this: an algorithm is superstabilizing if the following two conditions are satisfied. (1) It is self-stabilizing. (2) When it is started in a legitimate configuration and a topology change occurs, the *passage predicate* holds and continues to hold until the computation reaches a legitimate state. Here, a passage predicate is a global safety predicate for outputs of processes that a system maintains during convergence after a topology change in a legitimate configuration.

An *interrupt statement* is given in an algorithm description. When a topology change event occurs, the interrupt statement is atomically invoked with the event at a process on which the event is incident. Although the interrupt statement is executed locally by a process that detects a topology change event, a passage predicate is globally maintained.

2.3.2 Output stability

Output stable[7],[8] self-stabilization is one of the extensions of self-stabilization for dynamic networks in such a way that the number of output changes is limited, which contributes to stability of input of protocols in upper layers and stability of output for users. A self-stabilizing algorithm is output stable[7],[8] if and only if the number of changes of output variables of processes to stabilize again depends on the number of topology changes and processes hit by transient faults that occur in a legitimate configuration.

For output stability, the following performance measures are used. *Locally temporal instability* is the maximum (worst) number of times a process changes the value of its output variables after an external event until the system reaches a legitimate configuration. *Temporal instability* is the maximum (worst) number of times processes change the value of its output variables after an external event until the system reaches a legitimate configuration. *Spatial instability* is the maximum (worst) number of processes that changes the value of their output variables after an external event until the system reaches a legitimate configuration.

2.3.3 Safe convergence

A *safe converging*[9] self-stabilization is one of the extensions of self-stabilization in such a way that, starting from arbitrary initial configuration in Γ, any computation quickly reaches a configuration in $\Lambda_F(\subseteq \Gamma)$, and then, with keeping configurations are in Λ_F, a computation eventually reaches a configuration in $\Lambda_O(\subseteq \Lambda_F \subseteq \Gamma)$. Each configuration in Λ_F (resp. Λ_O) is called a *feasibly legitimate* (resp. *optimally legitimate*) configuration. A set of feasible configuration Λ_F specifies a *safety* during convergence; any configuration is guaranteed to be in Λ_F in a computation from Λ_F to Λ_O, i.e., safety is maintained during the period.

3. The proposed algorithm

In this section, we present a self-stabilizing distributed algorithm SSMCol. The formal description of the proposed algorithm is shown in **Fig. 1**.

3.1 The main idea

The algorithm is based on the following idea.

- Rule 1: If P_i finds that assignments of colors conflict with neighbors, P_i reassigns its colors in such a way that there is no color conflict with neighbors.
- Rule 2: If P_i finds that assignment of colors is different from an assignment computed from available colors, especially, more colors are available, P_i reassigns its colors in such a way that there is no color conflict.

Based on this idea, the algorithm offers self-stabilization property; color conflicts are eliminated soon, and then, each process assigns colors to eventually achieve a proper multicoloring. This idea is also the basis of safe convergence property of the algorithm; color conflicts are eliminated as soon as possible, and without introducing new color conflicts, a proper multicoloring is eventually achieved. Here, safety property is that there is no color conflicts, and a metric for optimization is the maximality of the number of colors assigned to each process.

3.2 Technical details

Each process P_i maintains a local variable $f_i(\subseteq C)$. Let $\gamma \in \Gamma$ be any configuration. By f_i^γ, we denote the value of f_i in configuration γ. We define a function f^γ in such a way that $f^\gamma(P_i) \equiv f_i^\gamma$ for each $P_i \in V$. When configuration γ is clear from context, we simply write f instead of f^γ, and write f_i instead of f_i^γ. Similarly, we denote by Y_i^γ the value of Y_i in γ, and we simply write Y_i instead of Y_i^γ if γ is clear from context.

As defined in Fig. 1, the color selection function $\mathit{Select}_i(X)$ for each P_i has the following property. Once a color set f_i is assigned, Select_i returns a color set that contains f_i as a subset as long as all the colors in f_i is available. These properties are enough for not only the closure property, but also the convergence property of the algorithm.

By Λ, we denote a set of proper multicoloring configurations with respect to *Select*. The set Λ is a set of legitimate configurations of the algorithm.

3.3 Proof of correctness and performance

We show that SSMCol is self-stabilizing with respect to Λ. For each $P_i \in V$, we say that P_i *is locally multicoloring* in configuration γ if and only if $\forall P_j \in N_i : (f_i^\gamma \cap f_j^\gamma = \emptyset)$. It is easy to see that the function f^γ is a multicoloring if and only if, for each $P_i \in V$, P_i is locally multicoloring in γ. It is also easy to see that the function f^γ is a proper multicoloring if and only if f^γ is a multicoloring and $|f_i^\gamma| = w(P_i)$ holds for each $P_i \in V$.

We say that a configuration $\gamma \in \Gamma$ *is a multicoloring* if and only if the function f^γ is a multicoloring. We say that a configuration $\gamma \in \Gamma$ *is a proper multicoloring* if and only if the function f^γ is a proper multicoloring. We say that a configuration $\gamma \in \Gamma$ is a *multicoloring with respect to function Select* if and only if γ is a multicoloring and $f_i^\gamma = \mathit{Select}_i(C \backslash Y_i^\gamma)$ holds for each $P_i \in V$. We say that a configuration $\gamma \in \Gamma$ is a *proper multicoloring with respect to Select* if and only if

```
Constant
  $C$ — A set of colors, $|C| \geq (\Delta+1)w^{\max}$, where $w^{\max} = \max_{P_i \in V} w(P_i)$.
  $N_i\ (\subseteq V)$ — A set of neighbors of $P_i$.
  $w_i\ (\equiv w(P_i))$ — The weight of $P_i$.
Local variable
  $f_i \subseteq C$ — Color assignment (a set of colors) of $P_i$.
Macro definition
  $Y_i \equiv \bigcup_{P_j \in N_i} f_j$ — A set of colors used by neighbors.
  $Select_i(X)$ Returns deterministically
    $X'(\subseteq X)$ such that $f_i \subseteq X'$ and $|X'| = w(P_i)$ if $(f_i \subseteq X) \wedge (|X| > w(P_i))$,
    $X'(\subseteq X)$ such that $f_i \subseteq X'$ and $|X'| = |X|$ if $(f_i \subseteq X) \wedge (|X| \leq w(P_i))$,
    $X'(\subseteq X)$ such that $|X'| = w(P_i)$ if $(f_i \not\subseteq X) \wedge (|X| > w(P_i))$, and
    $X$ otherwise.
Action
  Rule 1: Re-assign colors in case of a conflict.
    $(f_i \cap Y_i \neq \emptyset) \rightarrow$
      $f_i := Select_i(C \backslash Y_i)$;
  Rule 2: Assign more colors if possible.
    $(f_i \cap Y_i = \emptyset) \wedge (f_i \neq Select_i(C \backslash Y_i)) \rightarrow$
      $f_i := Select_i(C \backslash Y_i)$;
```

Fig. 1 Algorithm SSMCol, a description for each $P_i \in V$.

γ is a multicoloring with respect to *Select* and it is a proper multicoloring.

Next lemma shows sufficiency of the number of colors for color assignment in a local-greedy manner.

Lemma 1 For each P_i and any configuration $\gamma \in \Gamma$ such that $|f_i^{\gamma}(P_i)| \leq w(P_i)$ for each $P_i \in V$, $|C \backslash Y_i^{\gamma}| \geq w(P_i)$ and $|Select_i(C \backslash Y_i^{\gamma})| = w(P_i)$ hold if P_i invokes $Select_i$ in γ. $\square$

First, we show the closure property of the algorithm, i.e., the algorithm terminates if and only if a proper multicoloring is computed and colors are assigned by *Select i* for each P_i.

Lemma 2 For any configuration γ, no process is enabled in γ if and only if f^{γ} is a proper multicoloring with respect to *Select*. $\square$

Next, we show convergence of the algorithm. Convergence proceeds in three phases as follows.

- Phase 1: Starting from any configuration γ_0, any computation eventually reaches a configuration γ_1 in which f^{γ_1} is a multicoloring which may not be proper.
- Phase 2: Then, starting from γ_1, any computation eventually reaches a configuration γ_2 in which f^{γ_2} is a multicoloring and $|f^{\gamma_2}(P_i)| \leq w(P_i)$ for each P_i.
- Phase 3: Finally, starting from γ_2, any computation eventually reaches a configuration γ_3 in which f^{γ_3} is a proper multicoloring.

Theorem 1 SSMCol is a self-stabilizing algorithm for the distributed proper multicoloring problem with respect to Λ. $\square$

We show the performance, i.e., the convergence time, of SSMCol.

Theorem 2 Convergence time of SSMCol is $O(n^2 w^{\max})$ steps. $\square$

Theorem 3 Convergence time of SSMCol is $O(1)$ rounds. $\square$

4. On the number of colors

In this subsection, we discuss the behavior of SSMCol when the number of available colors is limited. Such a setting is realistic in a real environment.

We first show the necessary number of colors for local-greedy strategy.

Theorem 4 Any self-stabilizing algorithm for the distributed proper multicoloring problem based on a local-greedy strategy, $(\Delta+1)w^{\max}$ is necessary for the number of colors. $\square$

We show that the proposed algorithm SSMCol assigns colors as many as possible (up to $w(P_i)$ for each P_i) and converges, even if the number of colors is smaller than the above lower bound. In a practical setting, this is the case. Specifically, we discuss a case when $|C| < (\Delta+1)w^{\max}$, i.e., the total number of colors is less than the number that is required to achieve proper multicoloring by a local-greedy strategy.

We introduce the concept of a maximal multicoloring as follows.

Definition 5 For a vertex weighted network $G(V,E,w)$ and a set of colors C, a function f, which assigns to each $P_i \in V$ a subset $f(P_i) \subseteq C$, is a *maximal* multicoloring if and only if

- f is multicoloring,
- $\forall P_i \in V \ : \ |f(P_i)| \leq w(P_i)$, and
- $\forall P_i \in V \ : \ (|f(P_i)| < w(P_i)) \Rightarrow (C = f(P_i) \cup (\cup_{P_j \in N_i} f(P_j)))$ □

Definition 6 For a vertex weighted network $G(V, E, w)$ and a set of colors C, the *distributed maximal multicoloring problem* is a problem such that each $P_i \in V$ computes a multicolor assignment $f_i \subseteq C$ in such a way that the function f, where $f(P_i) = f_i$ for each $P_i \in V$, is a maximal multicoloring for $G(V, E, w)$ and C. □

By Λ_M, we denote a set of maximal multicoloring configurations with respect to *Select*. The set Λ_M is a set of legitimate configurations of the algorithm in the setting with small number of colors.

Theorem 5 SSMCol is a self-stabilizing algorithm for the distributed maximal multicoloring problem with respect to Λ_M, when $|C| < (\Delta + 1)w^{\max}$ □

Theorem 6 Convergence time of SSMCol to Λ_M is $O(n^2 w^{\max})$ steps, and $O(1)$ rounds. □

5. Dynamic networks

In this section, we consider dynamic change of a network such as changes of network topology and weight vector. For each weighted network $G = (V, E, w)$, by $\Lambda(V, E, w)$ (resp., $\Lambda_M(V, E, w)$), we denote a set of legitimate configurations of SSMCol for the proper (resp., maximal) multicoloring problem.

In the problem of the proper (resp., maximal) multicoloring, a *safety* property of an algorithm is that there is no color conflict, and a *liveness* property of an algorithm is that a proper (resp., maximal) multicoloring is eventually obtained.

Lemma 3 presented below states that, in any computation that starts from a multicoloring configuration, even if removal of processes and links and change of weight vector occur arbitrary times *continuously* or *intermittently* in the computation, *safety is always guaranteed.* If such events stops, by self-stabilization, any computation eventually reaches a proper (or maximal) multicoloring configuration, and hence liveness property is guaranteed.

The next two lemmas show a basic property of SSMCol that are used to show superstabilization and output stability of SSMCol.

```
Interrupt statement
        $f_i := Select_i(C \backslash Y_i)$;
```

Fig. 2 The interrupt action for SSMCol, for each $P_i \in V$.

Lemma 3 Let a weighted network G (resp., G') be $G = (V, E, w)$ (resp., $G' = (V', E', w')$, where $V' \subseteq V$ and $E' \subseteq E$. Assume $|C| \geq (\Delta + 1)w^{\max}$ holds, where Δ is the maximum degree in G and G'. Let γ be any multicoloring configuration on G, and assume that network changes from G to G' in γ. Starting from γ, any computation eventually reaches a configuration $\lambda' \in \Lambda(V', E', w')$, and, multicoloring is maintained in any configuration in the subsequent computation. □

In case the number of colors is small, we have the following result.

Lemma 4 Let a weighted network G (resp., G') be $G = (V, E, w)$ (resp., $G' = (V', E', w')$, where $V' \subseteq V$ and $E' \subseteq E$. Assume $|C| < (\Delta + 1)w^{\max}$ holds, where Δ is the maximum degree in G and G'. Let γ be any multicoloring configuration on G, and assume that network changes from G to G' in γ. Starting from γ, any computation eventually reaches a configuration $\lambda' \in \Lambda_M(V', E', w')$, and multicoloring is maintained in any configuration in the computation. □

In the following subsections, we show that during the convergence after a topology change in a legitimate configuration, SSMCol promises special adaptability: superstabilization, output stability, and safe convergence.

5.1 Superstabilization of SSMCol

In this subsection, we discuss superstabilization of SSMCol. **Fig. 2** presents the interrupt statement to make SSMCol superstabilizing. The interrupt statement is invoked immediately when a topology change event occurs. *Superstabilization time* is the maximum number of steps or rounds to bring a configuration legitimate by a topology change event that occurs in a legitimate configuration.

Theorem 7 SSMCol is superstabilizing for the proper and maximal multicoloring problem, and the superstabilizing time is $O(\Delta^2 w^{\max})$ steps and $O(1)$ rounds. □

5.2 Output stability of SSMCol

In this subsection, we discuss output stability of SSMCol.

Theorem 8 SSMCol is output stable for the proper and maximal multicoloring problem, and the locally temporal instability is $O(w^{\max})$, the temporal instability is $O(\Delta^2 w^{\max})$, and the spatial instability is $O(\Delta^2)$. $\Box$

Note that a color (channel) assigned to a process is canceled (preempted) at most once when an external event occurs because conflicting colors are canceled, and then, available colors are added as many as necessary or possible. Although the number of changes of the output variable f_i for each P_i by an external event (the local temporal instability) is $O(w^{\max})$, the effect to applications in the upper layer is very small.

5.3 Safe convergence of SSMCol

In this subsection, we discuss safe convergence of SSMCol. Let Λ_F be a set of multicoloring configurations. Let $\Lambda_O = \Lambda$ be a set of proper multicoloring configurations with respect to *Select*. Note that a relation $\Lambda_O \subseteq \Lambda_F \subseteq \Gamma$ holds.

Theorem 9 SSMCol is a self-stabilizing algorithm with safe convergence for the distributed proper multicoloring problem with respect to (Λ_F, Λ_O). Convergence time to Λ_F is $O(n^2 w^{\max})$ steps, and then, within $O(n^2 w^{\max})$ steps, computation reaches a configuration in Λ_O. $\Box$

From Theorem 5, we have the following corollary.

Corollary 1 SSMCol is a self-stabilizing algorithm with safe convergence with respect to (Λ_F, Λ_M). for the distributed maximal multicoloring problem for any color set C. $\Box$

6. Conclusion

In this paper, we proposed a distributed algorithm for the multicoloring problem. Our algorithm has self-stabilizing, superstabilizing, output stable, and safe converging properties that are useful in dynamic distributed systems. In addition, our distributed algorithm is a local algorithm in a sense that each process communicates only its direct neighboring processes, and hence no centralized control is necessary.

When the number of available colors are limited, some nodes may not be assigned any color because our algorithm is based on a local-greedy strategy. An extension of our algorithm to solve this problem by the idea of borrowing is a future task. Although we assumed a strong computational model (the state-reading model and the central daemon) in this paper, design of an algorithm assuming weaker computational model, such as the message passing model with probabilistic message loss, is also left as a future work.

Acknowledgments This work is supported in part by Grant-in-Aid for Scientific Research ((B) 19300012) of JSPS, Grand-in-Aid for Young Scientists ((Start-up) 21800031) of JSPS, and Kayamori Foundation of Informational Science Advancement.

References

1) Dijkstra, E.: Self-Stabilizing Systems in Spite of Distributed Control, *Communications of the ACM*, Vol.17, No.11, pp.643–644 (1974).
2) Ghosh, S. and Karaata, M.: A self-stabilizing algorithm for coloring planar graphs, *Distributed Computing*, Vol.7, pp.55–59 (1993).
3) Sur, S. and Srimani, P.: A self-stabilizing algorithm for coloring bipartite graphs, *Information Sciences*, Vol.69, pp.219–227 (1993).
4) Gradinariu, M. and Tixeuil, S.: Self-stabilizing Vertex Coloring of Arbitrary Graphs, *Proceedings of the 4th International Conference on Principles of Distributed Systems (OPODIS)*, pp.55–70 (2000).
5) Hedetniemi, S.T., Jacobs, D.P. and Srimani, P.K.: Linear time self-stabilizing colorings, *Information Processing Letters*, Vol.87, No.5, pp.251–255 (2003).
6) Dolev, S. and Herman, T.: Superstabilizing protocols for dynamic distributed systems, *Chicago Journal of Theoretical Computer Science*, Vol.3, No.4 (1997).
7) Yamauchi, Y., Ooshita, F., Kakugawa, H. and Masuzawa, T.: Output stability of self-stabilizing protocols against topology changes and transient faults, *Proceedings of the 8th International Conference on Applications and Principles of Information Science (APIS)* (2009).
8) Yamauchi, Y., Kamei, S., Ooshita, F., Kakugawa, H. and Masuzawa, T.: Output stability of self-stabilizing protocols against topology changes and transient faults, *In preparation for submission to a journal.*
9) Kakugawa, H. and Masuzawa, T.: A Self-Stabilizing Minimal Dominating Set Algorithm with Safe Convergence, *Proceedings of the 8th IPDPS Workshop on Advances in Parallel and Distributed Computational Models (APDCM)*, p.263 (2005).
10) Narayanan, L.: Channel Assignment and Graph Multicoloring, *Handbook of Wireless Networks and Mobile Computing*, John Wiley & Sons, Inc., New York, NY, USA, chapter4, pp.71–94 (2002).