algorithms

MDPI

Article

Computing Fault-Containment Times of Self-Stabilizing Algorithms Using Lumped Markov Chains †

Volker Turau

Institute of Telematics, Hamburg University of Technology, 21073 Hamburg, Germany; turau@tuhh.de

† An extended abstract of this work appeared at the 19th International Symposium on Stabilization, Safety, and Security of Distributed Systems 2017 in Boston, USA.

Received: 10 February 2018; Accepted: 2 May 2018; Published: 3 May 2018

Abstract: The analysis of self-stabilizing algorithms is often limited to the worst case stabilization time starting from an arbitrary state, i.e., a state resulting from a sequence of faults. Considering the fact that these algorithms are intended to provide fault tolerance in the long run, this is not the most relevant metric. A common situation is that a running system is an a legitimate state when hit by a single fault. This event has a much higher probability than multiple concurrent faults. Therefore, the worst case time to recover from a single fault is more relevant than the recovery time from a large number of faults. This paper presents techniques to derive upper bounds for the mean time to recover from a single fault for self-stabilizing algorithms based on Markov chains in combination with lumping. To illustrate the applicability of the techniques they are applied to a new self-stabilizing coloring algorithm.

Keywords: distributed algorithms; fault-tolerance; self-stabilization; Markov chain; lumping

1. Introduction

Fault tolerance aims at making distributed systems more reliable by enabling them to continue the provision of services in the presence of faults. The strongest form is *masking fault tolerance*, where a system continues to operate after faults without any observable impairment of functionality, i.e., safety is always guaranteed. In contrast *non-masking fault tolerance* does not ensure safety at all times. Users may experience incorrect system behavior, but eventually the system will fully recover. The potential of this concept lies in the fact that it can be used in cases where masking fault tolerance is too costly or even impossible to implement [1]. *Self-stabilizing algorithms* belong to the category of distributed algorithms that provide non-masking fault tolerance. They guarantee that systems eventually recover from transient faults of any scale such as perturbations of the state in memory or communication message corruption [2]. A critical issue is the length of the time span until full recovery. Examples are known where a memory corruption at a single process caused a vast disruption in large parts of the system and triggered a cascade of corrections to reestablish safety. Thus, an important issue for non-masking fault tolerance is the containment of the effect of faults.

A *fault-containing* system has the ability to contain the effects of transient faults in space and time. The goal is to keep the extent of disruption during recovery proportional to the extent of the faults. An extreme case of fault-containment with respect to space is given when the effect of faults is bounded to the set of faulty nodes. Azar et al. call this *error confinement* [3]. More relaxed forms of fault-containment are known as time-adaptive self-stabilization [4], scalable self-stabilization [5], strong stabilization [6], and 1-adaptive self-stabilization [7].

A configuration is called *k-faulty*, if in a legitimate configuration exactly k processes are hit by a fault (a configuration is called *legitimate* if it conforms with the specification). A large body of

research focuses on fault-containment for 1-faulty configurations. Two metrics have been introduced to quantify the containment behavior in the 1-faulty case: *contamination radius* and *containment time* [8,9]. A distributed algorithm $\mathcal{A}$ has contamination radius r if only nodes within the r-hop neighborhood of the faulty node change their state during recovery from a 1-faulty configuration. The containment time of $\mathcal{A}$ denotes the worst-case number of rounds any execution of $\mathcal{A}$ starting at a 1-faulty configuration needs to reach a legitimate configuration. In technical terms this corresponds to the *worst case time to recover* in case of a single fault. For randomized algorithms the expected number of rounds to reach a legitimate configuration corresponds to the *mean time to recover* (MTTR).

Over the last two decades a large number of self-stabilizing algorithms have been published. Surprisingly the analysis of the vast majority of these algorithms is confined to the worst case stabilization time starting from an arbitrary configuration. Considering the fact that these algorithms are intended to provide fault tolerance in the long run this is not the most relevant metric at all. From a practical point of view the recovery time from a 1-faulty configuration is more interesting. This statement is justified considering the fact that the probability for a 1-faulty configuration is much higher than that for a k-faulty configuration with a large value of k. The reason is that a distributed system consists of independently operating computers where transient faults such as memory faults in different computers are independent events. Considering this fact it comes as a surprise that most papers consider only arbitrary initial states (i.e., k-faulty configuration for any k) instead of focusing on 1-faulty configuration. Only in a few cases fault-containment metrics have been considered [10,11]. This is even more surprising considering the fact that the many techniques available to determine the worst case stabilization time of an algorithm, e.g., potential functions and convergence stairs, can also be used to compute the containment time.

This paper discusses a technique to analyze the containment time of randomized self-stabilizing algorithms with respect to memory and message corruption. The execution of the algorithm is modeled as a stochastic process. Let X be the random variable that represents the number of rounds the system requires to reach a legitimate configuration when started in a 1-faulty configuration. Then the MTTR of the algorithm is equal to $E[X]$, the expected value of X. Thus, we are interested in upper bounds for $E[X]$. Sometimes it is possible to derive an explicit expression for $E[X]$ or use results about absorbing Markov chains for this purpose. These equations may be solvable with a software package based on symbolic mathematics. However, the state space explosion problem precludes success for many real world problems. An important technique for the reduction of the complexity of Markov chains is *lumping* [12]. Lumping is a method based on the aggregation of states that exhibit the same or similar behavior. It leads to a smaller Markov chain that retains the same performance characteristics as the original one. But often lumping is not immediately applicable because the structure of the Markov chain is too complex. In some of these cases a weaker form of lumping can lead to Markov chains with a simpler structure that can still be used to derive an upper bound for the absorption time.

The contribution of this paper is a discourse about the containment time of self-stabilizing algorithms. We present and apply techniques based on Markov chains to compute upper bounds for this metric. In particular we demonstrate how lumping can be applied to reduce the complexity of the Markov chains. To demonstrate the usability of the techniques we apply it to a new self-stabilizing coloring algorithm as a case study. We derive an absolute bound for the expected containment time and show that the variance is bounded by a surprisingly small constant independent of the network's size. We believe that the techniques can also be applied to other algorithms.

2. Related Work

Self-stabilizing algorithms are analyzed with different techniques such as potential functions, convergence stairs, and Markov chains. The latter are particularly useful for randomized algorithms [13]. Their main drawback is that in order to set up the transition matrix the graph's adjacency matrix must be known. This restricts the applicability of this method to *small* or highly *symmetric* instances. DeVille et al. apply model checking tools to Markov chains for cases of networks of small size ($n \leq 7$) to determine the

expected stabilization time [14]. An example for highly symmetric networks are ring topologies [15,16]. Fribourg et al. model randomized distributed algorithms as Markov chains using the technique of coupling to compute upper bounds for the stabilization times [15]. Yamashita uses Markov chains to model self-stabilizing probabilistic algorithms and to prove stabilization [16]. Mitton et al. consider a randomized self-stabilizing $\Delta + 1$-coloring algorithm and model it in terms of urns/balls using a Markov chain to get a bound for the stabilization time [17]. Their evaluation is restricted to networks up to 1000 nodes. Crouzen et al. model faulty distributed algorithms as Markov decision processes to incorporate the effects of random faults when using a non-deterministic scheduler [18]. They used the PRISM model-checker to compute long-run average availabilities.

3. System Model

This paper uses the synchronous model of distributed computing as defined in the standard literature [2,8,19]. A distributed system is represented as an undirected graph $G(V, E)$ where V is the set of *nodes* and $E \subseteq V \times V$ is the set of *edges*. Let $n = |V|$ and $\Delta(G)$ denote the maximal degree of G. The topology is assumed to be fixed. If two nodes are connected by an edge, they are called *neighbors*. The set of neighbors of node v is denoted by $N(v) \subseteq V$ and $N[v] = N(v) \cup \{v\}$. Each node stores a set of variables. The values of all variables constitute the *local state* of a node. Let σ denote the set of possible local states of a node. The *configuration* of a system is the tuple of all local states of all nodes. $\Sigma = \sigma^n$ denotes the set of global states. A configuration is called *legitimate* if it conforms with the specification. The set of all legitimate configurations is denoted by $\mathcal{L}$.

Nodes communicate either via locally shared memory or by exchanging messages. In the shared memory model each node executes a protocol consisting of a list of rules of the form *guard* $\longrightarrow$ *statement*. The guard is a Boolean expression over the variables of the node and its neighbors. The statement consists of a series of commands. A node is called *enabled* if one of its guards evaluates to true. The execution of a statement is called a *move*.

In the message passing model a node performs three actions per round: receiving messages from neighbors, executing code, and sending messages to neighbors. Direct access to the state of neighboring nodes is impossible. Two nodes u and v communicate via two link registers: u writes in its register and v reads from it and v writes in its register and u reads from it. In this model the state of a node also includes the states of its registers. This works assumes the $\mathcal{CONGEST}$ model of distributed computation [19]. Algorithms in the $\mathcal{CONGEST}$ model enforce a $O(\log n)$ limitation on the maximum message size. Hence, with a single message only a constant number of node identifiers in the range $\{0, \ldots, n\}$ can be transmitted.

Execution of the statements is performed in a synchronous style, i.e., all enabled nodes execute their code in every round. An *execution* $e = \langle c_0, c_1, c_2, \ldots \rangle$, $c_i \in \Sigma$ is a sequence of configurations, where c_0 is called the *initial configuration* and c_i is the configuration after the i-th round. In other words, if the current configuration is c_{i-1} and all enabled nodes make a move, then this yields c_i.

Let $\mathcal{A}$ be a distributed algorithm and $\mathcal{L} \subseteq \Sigma$ a set of configurations. $\mathcal{A}$ is called *self-stabilizing* with respect to $\mathcal{L}$ if it satisfies the convergence and closure properties. The first property states that every execution of $\mathcal{A}$ reaches $\mathcal{L}$ after a finite number of rounds. The second property states that $\mathcal{A}(c) \in \mathcal{L}$ for all $c \in \mathcal{L}$ as long as no fault occurs. The worst case stabilization time $ST_{\mathcal{A}}(G)$ of $\mathcal{A}$ for a graph G is equal to the maximal number of rounds after which $\mathcal{A}$ reaches a legitimate configuration of G regardless of the initial configuration under the assumption that no errors occur.

Definition 1. *A configuration $c \in \Sigma$ of a self-stabilizing algorithm with respect to $\mathcal{L}$ is called k-faulty if a configuration $c' \in \Sigma$ satisfying $\mathcal{L}$ exists such that c differs from c' in the local states of at most k nodes.*

Note that for the message passing model this definition also covers message corruption. This paper analyzes the most common fault situation: 1-faulty configurations. They arise when a single node v is hit by a memory corruption or a single message sent by v is corrupted.

The containment behavior of a self-stabilizing algorithm is characterized by the contamination radius and the containment time.

Definition 2. *Let $\mathcal{A}$ be a self-stabilizing algorithm.*

1. *The containment time of $\mathcal{A}$ denotes the worst-case number of rounds any execution of $\mathcal{A}$ starting at a 1-faulty configuration needs to reach a legitimate configuration.*
2. *Let R_v be the subgraph induced by the nodes engaged in the recovery process from a 1-faulty configuration of $\mathcal{A}$ triggered by a fault at v. Then $\max\{dist(v,w) \mid w \in R_v\}$ is called the contamination radius.*

The stabilization time $ST_{\mathcal{A}}(R_v)$ is an obvious upper bound for the containment time.

4. Examples

Before presenting techniques to compute these metrics we give some examples using the shared memory model to illustrate the two definitions.

4.1. Contamination Radius

Consider an algorithm in the shared memory model with contamination radius r. A single fault will not spread beyond the r-hop neighborhood of the faulty node v. In this case $R_v \subseteq G_v^r$; G_v^r is the subgraph induced by nodes w with $dist(v,w) \leq r$. As an example consider the well known self-stabilizing algorithm $\mathcal{A}_1$ to compute a maximal independent set (see Algorithm 1). It uses a single variable *state*. A configuration is legitimate if nodes with $state = IN$ form a maximal independent set.

Algorithm 1: Self-stabilizing algorithm $\mathcal{A}_1$ to compute a MIS .

```
if state = IN ∧ ∃w ∈ N(v) s.t. w.state = IN then
  state := OUT
if state = OUT ∧ ∀w ∈ N(v) w.state = OUT then
  if random bit from 0,1 = 1 then
    state := IN
```

Lemma 1. *Algorithm $\mathcal{A}_1$ has contamination radius two.*

Proof. Consider a 1-faulty configuration where node v is hit by a memory corruption. First suppose the state of v changed from IN to OUT. Let $u \in N(v)$ then $u.state = OUT$. If u has an neighbor $w \neq v$ with $w.state = IN$ then u will not change its state during recovery. Otherwise, if all neighbors of u except v had state OUT node u may change state during recovery. But since these neighbors of u have a neighbor with state IN they will not change their state. Thus, in this case only the neighbors of v may change state during recovery.

Suppose that $v.state$ changed from OUT to IN. Then v and those neighbors of v with state IN can change to OUT. Then arguing as in the first case only nodes within distance two of v may change their state during recovery. □

Next we consider another example: $\Delta + 1$-coloring. Most distributed algorithms for this problem follow the same pattern. A node that realizes that it has selected the same color as one of its neighbors chooses a new color from a finite color palette. This palette does not include the current colors of the node's neighbors. To be executed under the synchronous scheduler these algorithms are either randomized or use identifiers for symmetry breaking. Variations of this idea are followed in [17,20,21]. As an example consider algorithm $\mathcal{A}_2$ from [20] (see Algorithm 2). $\mathcal{A}_2$ has a single variable c. A configuration is legitimate if the values of variable c describe a valid $\Delta + 1$-coloring.

Due to its choice of a new color from the palette algorithm $\mathcal{A}_2$ has contamination radius at least $\Delta(G)$ (see Figure 1).

Algorithm 2: Self-stabilizing $\Delta+1$-coloring algorithm $\mathcal{A}_2$ from [20].

if $c \neq \max(\{0,\ldots,\Delta\}\backslash\{w.c \mid w \in N(v)\})$ **then**
  if *random bit from 0,1 = 1* **then**
    $c := \max(\{0,\ldots,\Delta\}\backslash\{w.c \mid w \in N(v)\})$

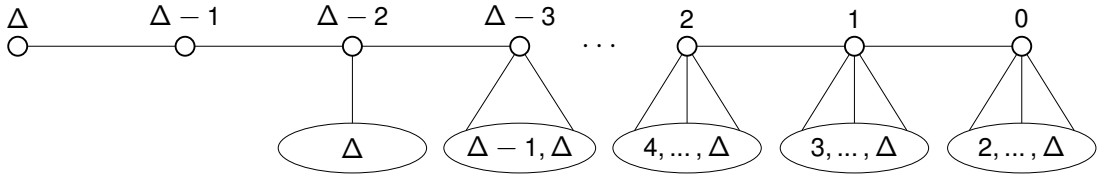


Figure 1. The numbers indicate the nodes' colors. If the left-most node is hit by a fault and changes its color to $\Delta-1$, then itself and its neighbor are enabled. With probability 0.5 the second node changes its color to Δ. This enables the third node which changes its color to $\Delta-1$ with probability 0.5. This may causes a cascade of changes in which all nodes on the horizontal line change color.

A minor modification of algorithm $\mathcal{A}_2$ dramatically changes matters. Algorithm $\mathcal{A}_3$ (see Algorithm 3) has contamination radius 1 (see Lemma 2). Note that neighbors of v that change their color during recovery form an independent set.

Algorithm 3: Self-stabilizing $\Delta+1$-coloring algorithm $\mathcal{A}_3$.

if $\exists w \in N(v)$ *s.t.* $c = w.c$ **then**
  if *random bit from 0,1 = 1* **then**
    $c :=$ choose $\{0,\ldots,\Delta\}\backslash\{w.c \mid w \in N(v)\}$

Lemma 2. *Algorithm $\mathcal{A}_3$ has contamination radius one.*

Proof. Consider a 1-faulty configuration where node v is hit by a memory corruption changing its color to a color c already chosen by at least one neighbor of v. Let $N_{\text{conflict}} = \{w \in N(v) \mid w.c = c\}$. In the next round the nodes in $N_{\text{conflict}} \cup \{v\}$ will get a chance to choose a new color. The choices will only lead to conflicts between v and other nodes in N_{conflict}. Thus, the fault will not spread beyond the set N_{conflict}. With a positive probability the set N_{conflict} will contain fewer nodes in each round. □

4.2. Containment Time

As the contamination radius the containment time strongly depends on the concrete structure of G. This can be illustrated with algorithm $\mathcal{A}_1$. Note that in this case R_v can contain any subgraph H with $\Delta(G)$ nodes. As an example let G consist of H and an additional node v connected to each node of H. A legitimate configuration is given if the state of v is IN and all other nodes have state OUT (Figure 2 left). If v changes its state to OUT due to a fault then all nodes may change to state IN during the next round. Thus, there is little hope for a bound below the trivial bound. Similar arguments hold for the second 1-faulty configuration of $\mathcal{A}_1$ shown on the right of Figure 2.

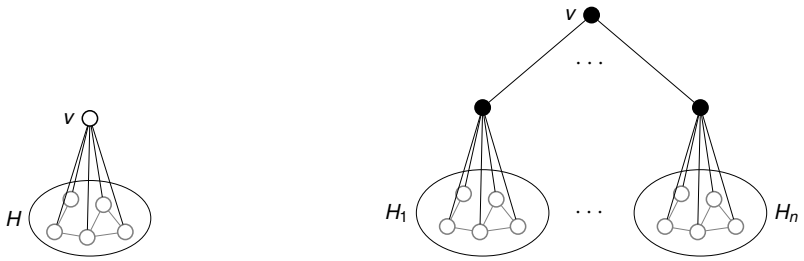


Figure 2. 1-faulty configurations of $\mathcal{A}_1$ caused by a memory corruption at v. Nodes drawn in bold have state IN. Subgraph H correspond to R_v. In the left graph, if node v changes to state OUT then all nodes in H are enabled, thus the worst case stabilization time is equal to that of subgraph H. In the right graph, if node v changes to IN, then v and its two neighboring nodes all change to OUT resulting in a configuration similar to the previous example.

5. Self-Stabilizing Algorithms and Markov Chains

A self-stabilizing algorithm $\mathcal{A}$ can be regarded as a transition systems of Σ. In each round the current configuration $c \in \Sigma$ is transformed into a new configuration $\mathcal{A}(c) \in \Sigma$. Each configuration of Σ occurs with a specific probability as the new configuration $\mathcal{A}(c)$. The source of randomness can have its origin in the scheduler—a probabilistic scheduler randomly selects enabled nodes to make a move—or in a random experiment within one of the rules. We assume that the scheduler and the random experiment are memory-less. Therefore, the execution of algorithm $\mathcal{A}$ can be described by a Markov chain with state set Σ and transition matrix $P = (p_{i,j})$, where $p_{ij} = Prob(\mathcal{A}(c_i) = c_j)$ gives the probability to move from configuration c_i to $c_j \in \Sigma$ in one round. This work uses the notation for Markov chains as introduced in [12]. For a self-stabilizing algorithm $\mathcal{A}$ this Markov chain is denoted by $\mathcal{C}_\mathcal{A}$. In the following we uses the terms configurations of A and states of $\mathcal{C}_\mathcal{A}$ as synonyms.

There is a close relation between the absorbing states of $\mathcal{C}_\mathcal{A}$ and the legitimate configurations $\mathcal{L}$ of $\mathcal{A}$. The closure property of a self-stabilizing algorithm guarantees that a configuration of $\mathcal{L}$ is always mapped to a configuration in $\mathcal{L}$, i.e., $\mathcal{A}(\mathcal{L}) \subseteq \mathcal{L}$. Whereas a state of a markov chain is called absorbing if it is impossible to leave this state. Note that a non-legitimate state of $\mathcal{A}$ cannot be an absorbing state of $\mathcal{C}_\mathcal{A}$. In silent self-stabilizing algorithms we have $\mathcal{A}(c) = c$ for all $c \in \mathcal{L}$. For non-silent algorithms we may without loss of generality also assume that $\mathcal{A}(c) = c$ for all $c \in \mathcal{L}$. If this condition is not given we can partition $\mathcal{L}$ into subsets $\mathcal{L}_1, \ldots, \mathcal{L}_s$ with $\mathcal{A}(\mathcal{L}_i) = \mathcal{L}_i$ and identify all configurations in $\mathcal{L}_i$ as corresponding to the same state. For the computation of the stabilization time this does not make a difference. Under this assumption the Markov chain $\mathcal{C}_\mathcal{A}$ is an absorbing Markov chain and the set of absorbing states corresponds to the set of legitimate states of $\mathcal{A}$. We can state the following easy to prove lemma.

Lemma 3. *Let c be a configuration of $\mathcal{C}_\mathcal{A}$. An absorbing state of $\mathcal{C}_\mathcal{A}$ is reached from c in expected B steps if and only if $\mathcal{A}$ stabilizes in expected B rounds from c.*

The computation of the expected time to absorption for an absorbing Markov chain using the transition matrix P is a simple matrix operation [12]. We assume a labeling of the states such that the t non-absorbing states come before the a absorbing states. Then P has the following canonical form

$$P = \begin{pmatrix} Q & R \\ 0 & E \end{pmatrix}$$

Here E is a $a \times a$ unit matrix and Q a $t \times t$ matrix. For an absorbing Markov chain, the matrix $N = (E - Q)^{-1}$ is called the *fundamental matrix* for P. Let t_i be the expected number of steps before the chain is absorbed, given that the chain starts in the i-th state, and let t be the column vector whose

i-th entry is t_i. Then $t = Nc$, where c is a column vector all of whose entries are 1. The variance of these numbers of steps is given by the entries of $(2N - E)t - t_{sq}$ where t_{sq} is derived from t by squaring each entry [12]. Thus, if N is the fundamental matrix of $\mathcal{C}_\mathcal{A}$ then the expected number of rounds after which algorithm $\mathcal{A}$ stabilizes is max Nc provided, the initial configuration is randomly chosen from the non-legitimate configurations.

There is still a big obstacles to practically apply this procedure. In order to compute the matrix N the probabilities $p_{i,j}$ need to be known explicitly. Without knowing the graph explicitly it is impossible to compute the probabilities $p_{i,j} = Prob(\mathcal{A}(c_i) = c_j)$ for all pairs of states c_i, c_j. This is only possible for small graphs or when the graph has a symmetric structure, e.g., a ring.

The fundamental matrix contains a lot of information which is not needed for the computation of the stabilization time. Therefore, a coarser analysis of the Markov chain would be sufficient. A common approach is to partition Σ into subsets $\Sigma_0, \ldots, \Sigma_l$ and consider these as the states of a new Markov chain (see Figure 3). The challenge is to define the transition probabilities of the new chain in a way that allows to transfer properties of this new chain to the original one.

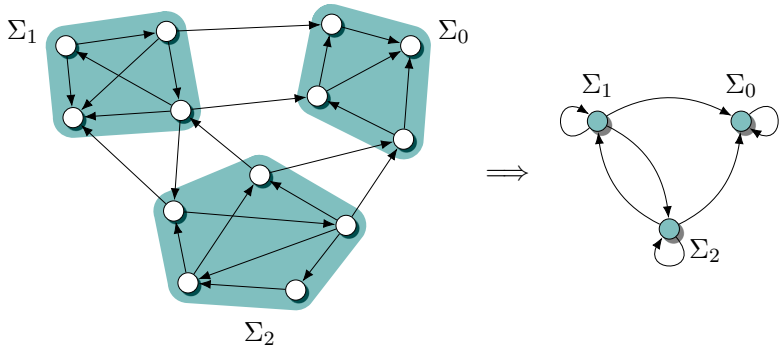


Figure 3. Reducing the number of states of a Markov chain.

A partition $P = \{\Sigma_0, \ldots, \Sigma_l\}$ of Σ is called *lumpable* if the subsets Σ_i have the property that for each pair i, j the probability of a state $c \in \Sigma_i$ to be transformed in one step into a state of Σ_j is independent of the choice of $c \in \Sigma_i$ (Definition 6.3.1 [12]). This probability is then interpreted as the transition probability from Σ_i to Σ_j. More formally, a Markov chain is *lumpable* with respect to partition $P = \{\Sigma_0, \ldots, \Sigma_l\}$ of Σ if for any $\Sigma_i, \Sigma_j \in P$ and any $c_1, c_2 \in \Sigma_i$

$$\sum_{c \in \Sigma_j} p(c_1, c) = \sum_{c \in \Sigma_j} p(c_2, c).$$

A lumpable Markov $\mathcal{C}$ chain defines a new Markov chain $\mathcal{C}^P$ with state set P and transition probabilities

$$p(\Sigma_i, \Sigma_j) = \sum_{c \in \Sigma_j} p(c_i, c).$$

The following result proved in [12] , p. 128.

Lemma 4. *Let $c \in \Sigma_i$ be a state of a lumpable Markov chain $\mathcal{C}$. Then the expected time to reach from c an absorbing state of $\mathcal{C}$ is equal to the expected time to reach from Σ_i an absorbing state of $\mathcal{C}^P$.*

The last two lemmas prove the following theorem.

Theorem 1. *Let $\mathcal{A}$ be a self-stabilizing algorithm with Σ the set of configurations. Let $P = \{\Sigma_0, \ldots, \Sigma_l\}$ be a partition of Σ with $\Sigma_0 = \mathcal{L}$ such that $\mathcal{C}_\mathcal{A}$ is lumpable with respect to P. For any i an absorbing state of ${\mathcal{C}_\mathcal{A}}^P$ is reached from Σ_i in expected B steps if and only if $\mathcal{A}$ stabilizes in expected B rounds starting in any $c \in \Sigma_i$.*

Unfortunately it is rather difficult to make use of Theorem 1 under general conditions. This situations changes when the theorem is used to compute the containment time of a self-stabilizing algorithm $\mathcal{A}$. Remember that the containment time of $\mathcal{A}$ denotes the worst-case number of rounds any execution of $\mathcal{A}$ starting in a 1-faulty configuration needs to reach a legitimate configuration. Thus, the containment time of $\mathcal{A}$ is $ST_{\mathcal{A}}(R_v)$, where v is the faulty node. There are two aspects that ease the application of Theorem 1: Either R_v has a symmetric structure or R_v is small.

To illustrate this approach we consider again algorithm $\mathcal{A}_3$. Let v be a node that changes in a legitimate state its color to c_f due to a memory fault (see Figure 4). Let c_0 be the new configuration. This causes a conflict with those neighbors of v that had chosen c_f as their color. After the fault only nodes contained in R_v (a star graph) change their state (see Figure 4 right). Once a neighbor has chosen a color different from c_f then it becomes passive (at least until the next transient fault).

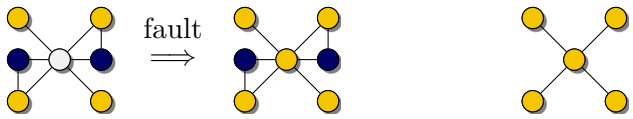


Figure 4. A 1-faulty configuration c_0 for algorithm $\mathcal{A}_3$ where node v was hit by a fault changing its color to c_f causing a conflict. The corresponding graph R_v is depicted at the right side.

The set Σ is equal to the set of all configurations of R_v reachable from c_0. To partition this set let d be the number of neighbors of v that have color c_f in c_0. Let $P = \{\Sigma_0, \ldots, \Sigma_d\}$ where Σ_j is the subset of Σ where exactly $d - j$ neighbors of v are in conflict with v. Then $\Sigma_0 = \{c_0\}$ and $\Sigma_d \subseteq \mathcal{L}$. Figure 5 shows some configurations belonging to the sets Σ_0, Σ_2, and Σ_3. Let $c \in \Sigma_i$. Then $\mathcal{A}_3(c) \notin \Sigma_j$ for all $j < i$. Unfortunately the partition P is not lumpable because the probability of a configuration $c \in \Sigma_i$ to be transformed in one round into a fixed configuration of Σ_j is not independent of the choice of $c \in \Sigma_i$. But even in these cases Theorem 1 can lead to an upper bound of the stabilization time. This is proved in the following theorem.

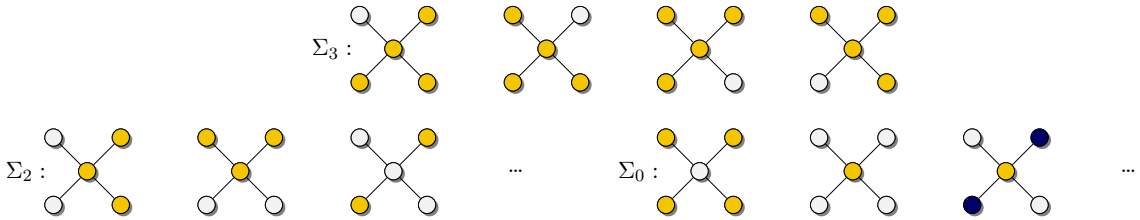


Figure 5. Elements of the partition of Σ for a 1-faulty configuration of algorithm $\mathcal{A}_3$ as described above. Σ_0 consists of legitimate configurations only.

Theorem 2. *Let $\mathcal{A}$ be a self-stabilizing algorithm with Σ the set of configurations. Let $P = \{\Sigma_0, \ldots, \Sigma_l\}$ be a partition of Σ with $\Sigma_0 = \mathcal{L}$ such that for all $i \geq 0$ if $c \in \Sigma_i$ then $\mathcal{A}(c) \in \Sigma_j$ with $j \geq i$. For $i < j$ let $c_{ij} \geq 0$ be a constant such that $Prob(\mathcal{A}(c) \in \Sigma_j) \geq c_{ij}$ for all $c \in \Sigma_i$. Furthermore, let $c_{ij} = 0$ for $j < i$ and $c_{ii} = 1 - \sum_{j=i+1}^{d} c_{ij}$ for $i = 0, \ldots, d$. Let $\mathcal{C}^{\mathcal{A}}$ be the Markov chain with states P and transition matrix (c_{ij}). If an absorbing state of $\mathcal{C}^{\mathcal{A}}$ is reached from Σ_i in expected B steps then the expected number of rounds $\mathcal{A}$ requires to stabilize starting in any $c \in \Sigma_i$ is at most B.*

Proof. Note

$$0 \leq \sum_{j=i}^{d} c_{ij} \leq \sum_{j=i}^{d} Prob(\mathcal{A}_3(c) \in \Sigma_j) = 1$$

for each fixed $c \in \Sigma_i$. Thus, $c_{ii} \geq 0$ and therefore the matrix (c_{ij}) is a stochastic with $c_{dd} = 1$ that describes the new Markov chain $\mathcal{C}^{\mathcal{A}}$. Remember that $\mathcal{C}_{\mathcal{A}}$ denotes the markov chain corresponding to algorithm $\mathcal{A}$. The difference between $\mathcal{C}_{\mathcal{A}}$ and $\mathcal{C}^{\mathcal{A}}$ is that with chain $\mathcal{C}^{\mathcal{A}}$ a node remains with a higher

probability in its current configuration instead of moving to a state Σ_i with lower index. Therefore, the expected number of steps of $\mathcal{C}^{\mathcal{A}}$ before being absorbed is an upper bound for the corresponding number in $\mathcal{C}_{\mathcal{A}}$. The choice of the probabilities implies that $\mathcal{C}^{\mathcal{A}}$ is lumpable for partition P. Hence, we can apply Theorem 1 to complete the proof. □

In the rest of this paper the introduced techniques including Theorem 2 are exemplary applied to a self-stabilizing $(\Delta + 1)$-coloring algorithm $\mathcal{A}_{col}$ using the message passing model.

6. Algorithm $\mathcal{A}_{col}$

This section introduces coloring algorithm $\mathcal{A}_{col}$ (see Algorithm 4). Computing a $\Delta + 1$-coloring in expected $O(\log n)$ rounds with a randomized algorithm is long known [22,23]. Algorithm $\mathcal{A}_{col}$ follows the pattern sketched in Section 4.1. We derived it from an existing algorithm (Algorithm 19 [24]) by adding the self-stabilization property. The presented techniques can also be applied to other randomized coloring algorithms such as [17,20,21]. The main difference is that $\mathcal{A}_{col}$ assumes the synchronous $\mathcal{CONGEST}$ message passing model. Algorithm $\mathcal{A}_{col}$ stabilizes after $O(\log n)$ rounds with high probability whereas the above cited self-stabilizing algorithms all require a linear number of rounds. Since synchronous local algorithms can be converted to asynchronous self-stabilizing algorithms [25], there are self-stabilizing $\Delta + 1$-coloring algorithms faster than $\mathcal{A}_{col}$. However, they entail a burden on memory resources and cause high traffic costs.

At the start of each round each node broadcasts its current color to its neighbors. Based on the information received from its neighbors a node decides either to keep its color (final choice), to choose a new color or no color (value $\perp$). In particular, with equal probability a node v draws uniformly at random a color from the set $\{0, 1, \ldots, \delta(v)\} \setminus tabu$ or indicates that it made no choice (see function `randomColor`). Here, *tabu* is the set of colors of neighbors of v that already made their final choice.

Algorithm 4: Algorithm $\mathcal{A}_{col}$ as executed by a node v in each round.

```
Set<Color> tabu := ∅, occupied := ∅;
broadcast(c, final) to all neighbors w ∈ N(v);
for all neighbors w ∈ N(v) do
    receive(c_w, final_w) from node w;
    if c_w ≠ ⊥ then
        occupied := occupied ∪ {c_w};
        if final_w then tabu := tabu ∪ {c_w} ;
if c = ⊥ ∨ c > δ(v) then
    final := false;
else
    if final then
        if c ∈ tabu then final := false ;
    else
        if c ∉ occupied then final := true ;
if final = false then c := randomColor(v, tabu) ;

function Color randomColor(Node v, Set<Color> tabu)
    if random bit from 0,1 = 1 then return ⊥ ;
    return random color from {0,1,...,δ(v)}\tabu;
```

In the original algorithm a node maintains a list with the colors of those neighbors that made their final choice. A fault changing this list is difficult to contain. Furthermore, in order to notice a memory corruption at a neighbor, each node must continuously send its state to all its neighbors and cannot

stop to do so. This is the price of self-stabilization and well known [2]. These considerations lead to the design of Algorithm $\mathcal{A}_{col}$. Each node only maintains the chosen color and whether its choice is final (variables `c` and `final`). In every round a node sends the values of `c` and `final` to all neighbors. $\mathcal{A}_{col}$ uses two additional variables `tabu` and `occupied`. They are reset at the beginning of every round. To improve fault containment a node's final choice of a color is only withdrawn if it coincides with the final choice of a neighbor. To achieve a $\Delta + 1$-coloring a node makes a new choice if its color is larger than its degree. This situation can only originate from a fault.

First we prove correctness and analyze the stabilization time of $\mathcal{A}_{col}$. A configuration is called a *legal coloring* if the values of variable c form a $\Delta + 1$-coloring. It is called *legitimate* if it is a legal coloring and $v.final = true$ for each node v.

Lemma 5. *A node v can change the value of variable $final$ from $true$ to $false$ only in the first round or when a fault occured just before the start of this round.*

Proof. Let $v.c = c_r$ at the beginning of the round. In order for v to set $v.final$ to $false$ one of the following conditions must be met at the start of the round: $c_r > \delta(v)$, $c_r = \perp$, or v has a neighbor w with $w.final = true$ and $w.c = c_r$.

The lemma is obviously true in the first case. Suppose that $c_r = \perp$ and $v.final = true$ at the round's start. If during the previous round $v.final$ was set to $true$ then $v.c$ can not be $\perp$ at the start of this round. Hence, at the start of the previous round $final$ already had value $true$. But in this case $v.c$ was not changed in the previous round and thus, $c_r \neq \perp$, contradiction. Finally assume the last condition. Then v and w cannot have changed their value of c in the previous round, because then $final = true$ would be impossible at the start of this round. Thus, v sent $(c_r, true)$ in the previous round. Hence, if $w.c = c_r$ at that time, w would have changed $w.final$ to $false$, again a contradiction. □

Lemma 6. *A node setting $final$ to $true$ will not change its variables as long as no error occurs.*

Proof. Let v be a node that executes $final := true$. If v changes the value back to $false$ in a later round then by Lemma 5 a fault must have occured. Thus in an error-free execution node v will never change variable $final$ again. Since a node can only change variable c if $final = false$ the proof is complete. □

Lemma 7. *If at the end of a round during which no error occured each node v satisfies $v.final = true$ then the configuration is legitimate and remains legitimate as long as no error occurs.*

Proof. Note that no node changed its color during that round. If at the start of the round $v.final = true$ was already satisfied then none of v's neighbors also having $final = true$ had the same color as v. Next consider a neighbor w of v with $w.final = false$ at the start of the round. Since v sent $(v.c, true)$ at the start of this round, node w would have set $final$ to $false$ if it had chosen the same color as v. Contradiction. Finally consider that case that $v.final = false$ at the start of the round. Since v changed $final$ to $true$, none of its neighbors had chosen the same color as v. Thus, the configuration is legitimate. Obviously, this property can only be changed by a fault. □

With these lemmas the next theorem is proved along the same lines as Lemma 10.3 in [24].

Theorem 3. *Algorithm $\mathcal{A}_{col}$ is self-stabilizing and computes a $\Delta + 1$-coloring within $O(\log n)$ rounds with high probability (i.e., with probability at least $1 - n^c$ for any $c \geq 1$). $\mathcal{A}_{col}$ has contamination radius 1.*

Proof. According to Lemma 7 it suffices to prove that all nodes terminate within $O(\log n)$ time with high probability. Let $v \in V$. Lemma 6 implies that the probability that v terminates in round $r > 1$ is equal to the probability that v sets $v.final$ to $true$ in round $r - 1$. This is the probability that v selects a color different from $\perp$ and from the selections of all neighbors that chose a value different from $\perp$ in round $r - 2$. Suppose that indeed $v.c \neq \perp$ at the end of round $r - 2$. Then $v.c \notin v.tabu$.

The probability that a given neighbor u of v selects the same color $u.c = v.c$ in this round is at most $1/2(\delta(v) + 1 - |v.tabu|)$. This is because the probability that u selects a color different from $\perp$ is $1/2$, and v has $\delta(v) + 1 - |v.tabu|$ different colors to select from. Since $r > 1$ all nodes in $v.tabu$ have $final = true$ and will never change this value. Thus, at most $\delta(v) - |tabu|$ neighbors select a new color. By the union bound, the probability that v selects the same color as a neighbor is at most

$$\frac{\delta(v) - |v.tabu|}{2(\delta(v) + 1 - |v.tabu|)} < \frac{1}{2}.$$

Thus, if v selects a color $v.c \neq \perp$, it is distinct from the colors of its neighbors with probability at least $1/2$. It holds that $v.c \neq \perp$ with probability $1/2$. Hence, v terminates with probability at least $1/4$.

The probability that a specific node v doesn't terminate within r rounds is at most $(3/4)^r$. By the union bound, the probability that there exists a vertex $v \in V$ that does not terminate within r rounds is at most $n(3/4)^r$. Hence, $\mathcal{A}_{col}$ terminates after $r = (c+1)4 \log n$ rounds, with probability at least $1 - n(3/4)^r \geq 1 - 1/n^c$ (note that $\log 4/3 > 1/4$). □

6.1. Fault Containment Time of Algorithm $\mathcal{A}_{col}$

There is a significant difference between the shared memory and the message passing model when analyzing the containment time. Firstly, a 1-faulty configuration also arises when a single message sent by a node v is corrupted. Secondly, this may cause v's neighbors to send messages they would not send in a legitimate configuration. Even though the states of nodes outside G_v^r do not change, these nodes may be forced to send messages. Thus, in general the analysis of the containment time cannot be performed by considering G_v^r only. This is only possible in cases when a fault at v does not force nodes at distance $r + 1$ to send messages they would not send had the fault not occurred.

In the following the fault containment behavior of $\mathcal{A}_{col}$ for 1-faulty configurations is analyzed. Two types of transient errors are considered:

1. A single broadcast message sent by v is corrupted. Note that the alternative of using $\delta(v)$ unicast messages instead a single broadcast has very good fault containment behavior but is slower due to the handling of acknowledgements.
2. Memory corruption at node v, i.e., the value of at least one of the two variables of v is corrupted.

The first case is analyzed analytically whereas for the second case Markov chains and lumping, (Theorem 2) are used. The *independent degree* $\delta_i(v)$ of a node v is the size of a maximum independent set of $N(v)$. Let $\Delta_i(G) = \max\{\delta_i(v) \mid v \in V\}$.

6.2. Message Corruption

If a message broadcast by v contains a color c_f different from $v.c$ or the value $false$ for variable $final$ then the message $(c_f, false)$ has no effect on any $w \in N(v)$ regardless of the value of c_f, since $w.final = true$ for all $w \in N(v)$. Thus, this corrupted message has no effect at all. In order to compute the containment time for $\mathcal{A}_{col}$ we first compute the contamination radius.

Lemma 8. *The contamination radius of algorithm $\mathcal{A}_{col}$ after a single corruption of a broadcast message sent by node v is* 1. *At most $\delta_i(v)$ nodes change their state during recovery.*

Proof. It suffices to consider the case that v broadcasts message $(c_f, true)$ with $c_f \neq v.c$. Let $N_{\text{conflict}}(v) = \{w \in N(v) \mid w.c = c_f\}$. The nodes in $N_{\text{conflict}}(v)$ form an independent set, because they all have the same color. Thus $|N_{\text{conflict}}(v)| \leq \delta_i(v)$.

Let $u \in V \backslash N[v]$. This node continues to send $(u.c, true)$ after the fault. Thus, a neighbor of u that changes its color will not change its color to $u.c$. This yields that no neighbor of u will ever send a message with $u.c$ as the first parameter. This is also true in case $u \in N(v) \backslash N_{\text{conflict}}(v)$. Hence, no node outside $N_{\text{conflict}}(v) \cup \{v\}$ will change its state, i.e., the contamination radius is 1.

Let $w \in N_{\text{conflict}}(v)$. When w receives the faulty message it sets $w.final$ to false. Before the faulty message was sent no neighbor of v had the same color as v. Thus, in the worst case a node $w \in N_{\text{conflict}}(v)$ will choose $v.c$ as its new color and send $(v.c, false)$ to all neighbors. Since $v.final = true$, this will not force state to change at v. Thus, v keeps broadcasting $(v.c, true)$ and no neighbor w of v will ever reach the state $w.c = v.c$ and $w.final = true$. Hence, v will never change its state. □

With this result Theorem 3 implies that the containment time of this fault is $O(\log \delta_i(v))$ on expectation. The following theorem gives a bound for the expected value of the containment time including its variance. Since the variance of a random variable is the expected value of the squared deviation from the mean, this theorem shows that the containment time does not deviate much from its expected value. A concrete bound can be obtained from this result using Chebyshev's inequality.

Theorem 4. *The expected containment time of algorithm $\mathcal{A}_{col}$ after a corruption of a message broadcast by node v is at most $\frac{1}{\ln 2} H_{\delta_i(v)} + 1/2$ rounds (H_i is the i^{th} harmonic number) with a variance of at most*

$$\frac{1}{\ln^2 2} \sum_{i=1}^{\delta_i(v)} \frac{1}{i^2} + \frac{1}{4} \leq \frac{\pi^2}{6 \ln^2 2} + \frac{1}{4} \approx 3.6737.$$

Proof. After receiving message $(c_f, true)$ all nodes $w \in N_{\text{conflict}}(v)$ set $w.final$ to $false$ and with equal probability $w.c$ to $\perp$ or to a random color $c_w \in \{0, 1, \ldots, \delta(w)\} \backslash w.tabu$. Note that $|w.tabu| \leq \delta(w)$ because $w.tabu = \{u.c \mid u \in N(w) \backslash v\} \cup \{c_f\}$. At the end of the round during which the corrupted message was received, node w can choose v's current color, because it may not be contained in the set $tabu$. This can not happen in the following rounds. Thus, if w chooses a color different from $\perp$ in the following rounds then this color is different from the colors of all of w's neighbors. Also in this case w will terminate after the following round because then it will set $final$ to $true$. Thus, after one round w has chosen a color that is different from the colors of all neighbors with probability at least $1/2$. Furthermore, this color will not change again. After one additional round w reaches a legitimate state.

Let the random variable X_d with $d = |N_{\text{conflict}}(v)|$ denote the number of rounds until the system has reached a legal coloring. For $w \in N_{\text{conflict}}(v)$ let Y_w be the random variable denoting the number of rounds until w has a legal coloring. By Lemma 8 $X_d = \max\{Y_w \mid w \in N_{\text{conflict}}(v)\}$. For $i \geq 1$ let $G(i) = Prob(X_d \leq i) = Prob(\max\{Y_w \mid w \in N_{\text{conflict}}(v)\} \leq i)$. Since the random variables Y_w's are independent $G(i) = Prob(X \leq i)^{|N_{\text{conflict}}(v)|}$ where X is a geometric random variable with $p = 0.5$. Thus $G(0) = 0$ and

$$G(i) = \left(\sum_{j=1}^{i} p(1-p)^{j-1} \right)^d.$$

Then $E[X_d] = \sum_{i=1}^{\infty} i Prob(X_d = i)$. Let $q = 1 - p$. Now for $i \geq 1$

$$Prob(X_d = i) = G(i) - G(i-1) = (1 - q^i)^d - (1 - q^{i-1})^d$$

$$= \sum_{j=0}^{d} \binom{d}{j} (-1)^{j+1} (1 - q^j) q^{j(i-1)} = \sum_{j=1}^{d} \binom{d}{j} \frac{(-1)^{j+1}(1-q^j)}{q^j} (q^j)^i.$$

This implies

$$E[X_d] = \sum_{j=1}^{d} \binom{d}{j} \frac{(-1)^{j+1}(1-q^j)}{q^j} \sum_{i=1}^{\infty} i (q^j)^i = \sum_{j=1}^{d} \binom{d}{j} \frac{(-1)^{j+1}}{(1-q^j)} = \sum_{j=1}^{d} \binom{d}{j} (-1)^{j+1} \sum_{l=0}^{\infty} (q^j)^l$$

$$= \sum_{l=0}^{\infty} \sum_{j=1}^{d} \binom{d}{j} (-1)^{j+1} (q^l)^j = \sum_{l=0}^{\infty} \left(1 + \sum_{j=0}^{d} \binom{d}{j} (-1)^{j+1} (q^l)^j \right) = \sum_{l=0}^{\infty} (1 - (1-q^l)^d).$$

The result follows from Lemma 9. The expression for the variance is proved in Lemma 10. □

Lemma 9. *For fixed* $0 < q < 1$ *and fixed* $d \geq 1$

$$\sum_{l=0}^{\infty}(1-(1-q^l)^d) \in [-\frac{1}{\ln q}H_d, -\frac{1}{\ln q}H_d + 1] \text{ and } \sum_{l=0}^{\infty}(1-(1-q^l)^d) \approx -\frac{1}{\ln q}H_d + \frac{1}{2}.$$

Proof. The function $f(x) = 1-(1-q^x)^d$ is for fixed values of d decreasing for $x \geq 0$. Furthermore, $f(0) = 1$. Hence,

$$\sum_{l=0}^{\infty}(1-(1-q^l)^d) \geq \int_0^{\infty} f(x)dx \geq \sum_{l=0}^{\infty}(1-(1-q^l)^d) - 1.$$

Using the substitution $u = 1-q^x$ the integral becomes

$$-\frac{1}{\ln q}\int_0^{\infty}\frac{1-u^d}{1-u}du = -\frac{1}{\ln q}\int_0^1\sum_{i=0}^{d-1}u^i du = -\frac{1}{\ln q}\sum_{i=1}^{d}\frac{1}{i} = -\frac{1}{\ln q}H_d.$$

Approximating $\int_i^{i+1} f(x)dx$ with $(f(i)+f(i+1))/2$ yields

$$\sum_{l=0}^{\infty}(1-(1-q^l)^d) \approx \int_0^{\infty} f(x)dx + \frac{f(0)}{2} = -\frac{1}{\ln q}H_d + \frac{1}{2}$$

□

Lemma 10. *For* $d > 0$ *the variance of the containment time is at most*

$$Var[X_d] = \frac{1}{\ln^2 2}\sum_{i=1}^{d}\frac{1}{i^2} + \frac{1}{4} \leq \frac{\pi^2}{6\ln^2 2} + \frac{1}{4} \approx 3.6737.$$

Proof.

$$Var[X_d] = E[X_d^2] - E[X_d]^2 = \sum_{i=1}^{\infty} i^2 Prob(X=i) - E[X_d]^2$$

By Lemma 11

$$\sum_{i=1}^{\infty} i^2 Prob(X=i) = \sum_{l=1}^{\infty}(2l+1)(1-(1-q^l)^d) = 2\sum_{l=1}^{\infty} l(1-(1-q^l)^d) + E[X_d]$$

Now Lemma 12 and Lemma 4 yield

$$\begin{aligned} Var[X_d] &\approx \frac{2}{\ln^2 2}\sum_{i=1}^{d}\frac{H_i}{i} + E[X_d] - E[X_d]^2 \approx \frac{2}{\ln^2 2}\sum_{i=1}^{d}\frac{H_i}{i} + \frac{H_d}{\ln 2} + \frac{1}{2} - \left(\frac{H_d}{\ln 2} + \frac{1}{2}\right)^2 \\ &= \frac{1}{\ln^2 2}\left(2\sum_{i=1}^{d}\frac{H_i}{i} - H_d^2\right) + \frac{1}{4} \end{aligned}$$

$$\begin{aligned} 2\sum_{i=1}^{d}\frac{H_i}{i} - H_d^2 &= 2\sum_{i=1}^{d}\sum_{j=1}^{i}\frac{1}{ij} - (1+\frac{1}{2}+\cdots+\frac{1}{d})^2 \\ &= 2\sum_{i=1}^{d}\frac{1}{i^2} + 2\sum_{i=1}^{d}\sum_{j=1}^{i-1}\frac{1}{ij} - 2\sum_{i=1}^{d}\sum_{j=1}^{i-1}\frac{1}{ij} - \sum_{i=1}^{d}\frac{1}{d^2} = \sum_{i=1}^{d}\frac{1}{i^2} = \frac{\pi^2}{6} \end{aligned}$$

□

Lemma 11. *Let $d > 0$, $q \in (0,1)$ and $Prob(X = i) = \sum_{j=1}^{d} \binom{d}{j} \frac{(-1)^{j+1}(1-q^j)}{q^j} (q^j)^i$. Then*

$$\sum_{i=1}^{\infty} i^2 Prob(X = i) = \sum_{l=1}^{\infty} (2l+1)(1-(1-q^l)^d)$$

Proof.

$$\begin{aligned}
\sum_{i=1}^{\infty} i^2 Prob(X = i) &= \sum_{j=1}^{d} \binom{d}{j} \frac{(-1)^{j+1}(1-q^j)}{q^j} \sum_{i=1}^{\infty} i^2 (q^j)^i \\
&= \sum_{j=1}^{d} \binom{d}{j} \frac{(-1)^{j+1}(1-q^j)}{q^j} \left(\frac{2q^{2j}}{(1-q^j)^3} + \frac{q^j}{(1-q^j)^2} \right) \\
&= \sum_{j=1}^{d} \binom{d}{j} (-1)^{j+1} \left(\frac{2q^j}{(1-q^j)^2} + \frac{1}{1-q^j} \right) \\
&= \sum_{j=1}^{d} \binom{d}{j} (-1)^{j+1} \sum_{l=0}^{\infty} (2l+1)(q^j)^l \\
&= \sum_{l=1}^{\infty} (2l+1) \sum_{j=1}^{d} \binom{d}{j} (-1)^{j+1} (q^l)^j \\
&= \sum_{l=1}^{\infty} (2l+1)(1-(1-q^l)^d)
\end{aligned}$$

For the first equation we refer to the proof of Theorem 4. The second equality makes use of

$$\sum_{i=1}^{\infty} i^2 x^i = \frac{2x^2}{(1-x)^3} + \frac{x}{(1-x)^2}$$

and the fourth equality uses the following two identities

$$\sum_{l=0}^{\infty} x^l = \frac{1}{(1-x)} \text{ and } \sum_{l=0}^{\infty} l x^l = \frac{x}{(1-x)^2}.$$

□

Lemma 12. *Let $d > 0$ and $q \in (0,1)$ then*

$$\sum_{l=1}^{\infty} l(1-(1-q^l)^d) \leq \frac{1}{(\ln q)^2} \sum_{i=1}^{d} \frac{H_i}{i}$$

Proof. We approximate $\sum_{l=1}^{\infty} l(1-(1-q^l)^d)$ with $\int_0^{\infty} x(1-(1-q^x)^d)dx$. Note that $x(1-(1-q^x)^d)$ has a single local maximum in the interval $[0,\infty)$. If the local maximum is within the interval $[y, y+1]$ with $y \in \mathbb{N}$ then the error is

$$\int_y^{y+1} x(1-(1-q^x)^d)dx.$$

This leads to a small overestimation of the sum as Figure 6b shows.

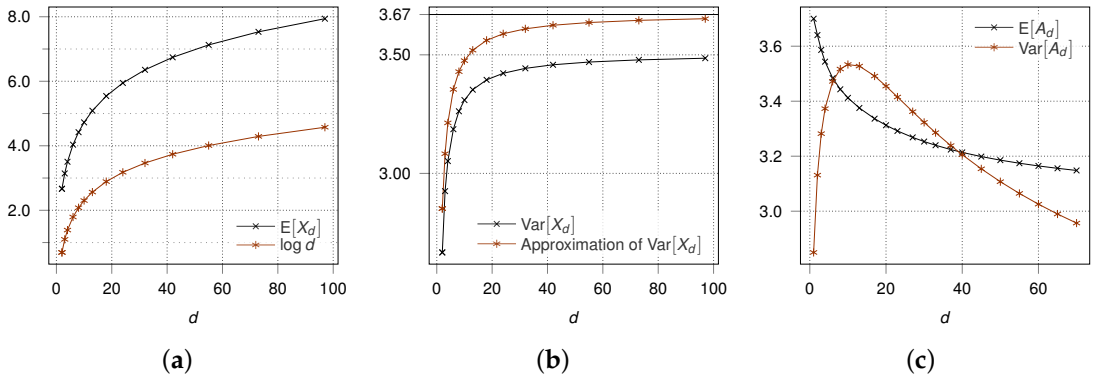
$$\begin{aligned}\sum_{l=1}^{\infty} l(1-(1-q^l)^d) &\leq \int_0^{\infty} x(1-(1-q^x)^d)dx \\ &= \frac{-1}{(\ln q)^2}\int_0^1 \ln(1-u)\frac{(1-u^d)}{1-u}du \\ &= \frac{-1}{(\ln q)^2}\sum_{i=0}^{d-1}\int_0^1 \ln(1-u)u^i du \\ &= \frac{1}{(\ln q)^2}\sum_{i=1}^{d}\frac{H_i}{i}.\end{aligned}$$

The first equation uses the substitution $u = 1-q^x$. The final result is based on the following identity

$$\int_0^1 \ln(1-u)u^i du = \frac{-H_{i+1}}{i+1}.$$

□

Theorem 4 gives an upper bound for the containment time and its variance of algorithm $\mathcal{A}_{col}$. To evaluate the quality of these upper bounds we modeled the behavior of this fault situation as a Markov chain and computed $E[X_d]$ and $Var[X_d]$ using a software package based on symbolic mathematics. Using Theorem 3.3.5 from [12] these computations showed that $\frac{1}{\ln 2}H_d + 1/2$ matches very well with $E[X_d]$ and that $E[X_d] \approx 2\log d$ (see Figure 6a). Furthermore, the gap between $Var[X_d]$ and the bound given in Theorem 4 is less than 0.2 (see Figure 6b).



(**a**) (**b**) (**c**)

Figure 6. Comparisons of computed with approximated values from Theorems 4 and 15. (**a**) Comparison of computed value of $E[X_d]$ with $\log d$ (Theorem 4); (**b**) Comparison of computed value of $Var[X_d]$ with approximation (Theorem 4); (**c**) $E[A_d]$ and $Var[A_d]$ from Lemma 15.

6.3. Memory Corruption

This section demonstrates the use of Markov chains and the application of Theorem 2. We consider the case that the memory of a single node v is hit by a fault. The analysis breaks down the stabilizing executions into several states and then computes the expected time for each of these phases. First we look at the case that the fault causes variable $v.final$ to change to $false$. If $v.c$ does not change, then a legitimate configuration is reached after one round. So assume $v.c$ also changes. Then the fault will not affect other nodes. This is because no $w \in N(v)$ will change its value of $w.c$ since $w.final = true$ and $v.final = false$. Thus, with probability at least $1/2$ node v will choose in the next round a color different from the colors of all neighbors and terminate one round later. Similar to X_d let random

variable Z_d denote the number of rounds until a legal coloring is reached ($d = |N_{\text{conflict}}(v)|$). It is easy to verify that $E[Z_d] = 3$ in this case.

The last case is that only variable $v.c$ is affected (i.e., $v.final$ remains *true*). The main difference to the case of a corrupted message is that this fault persists until $v.c$ has again a legitimate value. Let c_f be the corrupted value of $v.c$ and suppose that $N_{\text{conflict}}(v) = \{w \in N(v) \mid w.c = c_f\} \neq \emptyset$. A node outside $S = N_{\text{conflict}}(v) \cup \{v\}$ will not change its state (cf. Lemma 8). Thus, the contamination radius is 1 and at most $\delta_i(v) + 1$ nodes change state. Let $d = |N_{\text{conflict}}(v)|$. The subgraph G_S induced by S is a star graph with $d + 1$ nodes and center v.

Lemma 13. *To find a lower bound for $E[Z_d]$ we may assume that w can choose a color from $\{0,1\}\backslash tabu$ with $tabu = \emptyset$ if $v.final = false$ and $tabu = \{v.c\}$ otherwise and v can choose a color from $\{0,1,\ldots,d\}\backslash tabu$ with $tabu \subseteq \{0,1\}$.*

Proof. When a node $u \in S$ chooses a color with function `randomColor` the color is randomly selected from $C_u = \{0,1,\ldots,\delta(v)\}\backslash tabu$. Thus, if w and v choose colors in the same round, the probability that the chosen colors coincide is $|C_w \cap C_v|/|C_w||C_v|$. This value is maximal if $|C_w \cap C_v|$ is maximal and $|C_w||C_v|$ is minimal. This is achieved when $C_w \subseteq C_v$ and C_v is minimal (independent of the size of C_w) or vice versa. Thus, without loss of generality we can assume that $C_w \subseteq C_v$ and both sets are minimal. Thus, for $w \in N_{\text{conflict}}(v)$ the nodes in $N(w)\backslash\{v\}$ already use all colors from $\{0,1,\ldots,\delta(v)\}$ but 0 and 1 and all nodes in $N(v)\backslash N_{\text{conflict}}(v)$ already use all colors from $\{0,1,\ldots,\delta(v)\}$ but $0,1,\ldots,d$. Hence, a node $w \in N_{\text{conflict}}(v)$ can choose a color from $\{0,1\}\backslash tabu$ with $tabu = \emptyset$ if $v.final = false$ and $tabu = \{v.c\}$ otherwise. Furthermore, v can choose a color from $\{0,1,\ldots,d\}\backslash tabu$ with $tabu \subseteq \{0,1\}$. In this case $tabu = \emptyset$ if $w.final = false$ for all $w \in N_{\text{conflict}}(v)$. □

Thus, in order to bound the expected number of rounds to reach a legitimate state after a memory corruption we can assume that $G = G_S$ and $u.final = true$ and $u.c = 0$ (i.e., $c_f = 0$) for all $u \in S$. After one round $u.final = false$ for all $u \in S$. To apply Theorem 2 the set Σ of all configurations is partitioned into $d + 4$ subsets as follows:

- I: Represents the faulty state with $u.c = 0$ and $u.final = true$ for all $u \in S$.
- C^i: Node v and exactly $d - i$ non-center nodes will not be in a legitimate state after the following round ($0 \leq i \leq d$). In particular $v.final = false$ and $w.c = v.c \neq \perp$ or $v.c = w.c = \perp$ for exactly $d - i$ non-center nodes w.
- P: Node v has not reached a legitimate state but will do so in the next round. In particular $v.final = false$ and $v.c \neq w.c$ for all non-center nodes w.
- F: Node v is in a legitimate state, i.e., $v.final = true$ and $v.c \neq w.c$ for all non-center nodes w, but $w.c$ may be equal to $\perp$.

Note that I is the initial and F the absorbing state of the lumped Markov chain. Also when the system is in state F, then it is not necessarily in a legitimate state. This state reflects the set of configurations considered in the last section.

Lemma 14. *Table 1 describes the transition probabilities of the lumped Markov chain.*

Table 1. This table summarizes the probabilities for all transitions.

Number	Transition	Probability
1	$I \longrightarrow P$	$\frac{d-1}{2d} + \frac{1}{d}\left(\frac{1}{2}\right)^{d+1}$
2	$I \longrightarrow C^0$	$\frac{d-1}{d}\left(\frac{1}{2}\right)^{d+1} + \frac{1}{2d}$
3	$I \longrightarrow C^j$	$\binom{d}{d-j}\left(\frac{1}{2}\right)^{d+1}$ $(0 < j \leq d)$
4	$C^i \longrightarrow C^j$	$\binom{d-i}{d-j}\left(\frac{1}{2}\right)^{d-i+1} + \frac{1}{d-i+1}\binom{d-i}{j-i}\left(\frac{1}{4}\right)^{d-i}(3^{d-j} - 2^{d-j})$ $(0 \leq i \leq j \leq d)$
5	$C^i \longrightarrow P$	$\frac{1}{d-i+1}\left(\frac{3}{4}\right)^{d-i} + \frac{d-i-1}{2(d-i+1)}$ $(0 \leq i < d)$
6	$C^d \longrightarrow P$	$1/2$
7	$P \longrightarrow F$	1

Proof. We consider each case separately. The last two cases are trivial.

1. Note that $u.final = true$ and $u.c = 0$ for all $u \in S$.
   Case 0: $v.c = \perp$. Impossible.
   Case 1: $v.c = 0$. Impossible, since non-center nodes have $c = 0$ and $final = true$.
   Case 2: $v.c = 1$. This happens with probability $1/2d$. All non-center nodes w choose $w.c = \perp$, this happens with probability $1/2^d$.
   Case 3: $v.c > 1$. This happens with probability $(d-1)/(2d)$. Non-center nodes can make any choice. This gives the total probability for this transition as $(d-1)/(2d) + 1/(d2^{d+1})$.
2. Note that $u.final = true$ and $u.c = 0$ for all $u \in S$.
   Case 0: $v.c = \perp$. Non-center nodes choose $c = \perp$. The case has probability $1/2^{d+1}$
   Case 1: $v.c = 0$. Impossible (see transition $I \longrightarrow P$).
   Case 2: $v.c = 1$. At least one non-center nodes w choose $w.c = 1$, all others choose $w.c = \perp$. This case has probability $\frac{1}{2d}\sum_{l=1}^{d}\binom{d}{l}\left(\frac{1}{2}\right)^d = \frac{1}{2d}\left(\frac{1}{2}\right)^d(2^d - 1)$.
   Case 3: $v.c > 1$. This case is impossible.
3. Note that $u.final = true$ for all $u \in S$.
   Case 1: $v.c = 0$. This happens with probability $1/2(d-i+1)$. None of the $d-i$ non-center nodes w sets $w.c = 0$, this has probability $\left(\frac{3}{4}\right)^{d-i}$.
   Case 2: $v.c = 1$. Similar to case 1.
   Case 3: $v.c > 1$. (requires $d - i > 1$). This happens with probability $\frac{d-i-1}{2(d-i+1)}$. Non-center nodes can make any choice.
4. Note that $u.final = false$ for all $u \in S$.
   Case 1: $v.c = \perp$. This happens with probability $\frac{1}{2}$. $d-j$ non-center nodes choose $c = \perp$ (with probability $1/2^{d-j}$), the other $j-i$ non-center nodes choose $c \neq \perp$ (with probability $1/2^{j-i}$). The total probability for this case is $\binom{d-i}{d-j}\left(\frac{1}{2}\right)^{d-i+1}$.
   Case 2: $v.c = 0$. This happens with probability $1/(2(d-i+1))$. Exactly $j-i$ non-center nodes choose $c = 1$ (with probability $1/4^{j-i}$), $1 \leq l \leq d-j$ non-center nodes choose $c = 0$

(with probability $1/4^l$) and all other non-center nodes choose $c = \perp$ (with probability $1/2^{d-j-l}$). The total probability for this case is

$$\begin{aligned}
& \tfrac{1}{2(d-i+1)}\binom{d-i}{j-i}\left(\tfrac{1}{4}\right)^{j-i}\sum_{l=1}^{d-j}\binom{d-j}{l}\left(\tfrac{1}{4}\right)^{l}\left(\tfrac{1}{2}\right)^{d-j-l} \\
= & \tfrac{1}{2(d-i+1)}\binom{d-i}{j-i}\left(\tfrac{1}{4}\right)^{j-i}\sum_{l=1}^{d-j}\binom{d-j}{l}\left(\tfrac{1}{2}\right)^{d-j+l} \\
= & \tfrac{1}{2(d-i+1)}\binom{d-i}{j-i}\left(\tfrac{1}{4}\right)^{j-i}\left(\tfrac{1}{2}\right)^{d-j}\sum_{l=1}^{d-j}\binom{d-j}{l}\left(\tfrac{1}{2}\right)^{l} \\
= & \tfrac{1}{2(d-i+1)}\binom{d-i}{j-i}\left(\tfrac{1}{2}\right)^{d+j-2i}\sum_{l=1}^{d-j}\binom{d-j}{l}\left(\tfrac{1}{2}\right)^{l} \\
= & \tfrac{1}{2(d-i+1)}\binom{d-i}{j-i}\left(\tfrac{1}{2}\right)^{d+j-2i}\left(\left(\tfrac{3}{2}\right)^{d-j}-1\right) \\
= & \tfrac{1}{2(d-i+1)}\binom{d-i}{j-i}\left(\tfrac{1}{4}\right)^{d-i}\left(3^{d-j}-2^{d-j}\right)
\end{aligned}$$

Case 2: $v.c = 1$. Similar to Case 1.
Case 3: $v.c > 0$. This does not lead to C^j but to P.

5. Note that $u.final = false$ for all $u \in S$.
Case 1: $v.c = 0$. This happens with probability $1/2(d-i+1)$. None of the $d-i$ non-center nodes w sets $w.c = 0$, this has probability $(3/4)^{d-i}$.
Case 2: $v.c = 1$. Similar to case 1.
Case 3: $v.c > 1$. This happens with probability $\frac{d-i-1}{2(d-i+1)}$. Note $d > i$. Non-center nodes can make any choice.

□

We first calculate the expected number $E[A_d]$ of rounds to reach the absorbing state F. With Theorem 4 this will enable us to compute the expected number $E[Z_d]$ of rounds required to reach a legitimate system state. To build the transition matrix of the lumped Markov chain the $d+4$ states are ordered as $I, C^0, C^1, \ldots, C^d, P, F$. Let Q be the $(d+3) \times (d+3)$ upper left submatrix of P. For $s = -1, 0, 1, \ldots, d+1$ denote by Q_s the $(s+2) \times (s+2)$ lower right submatrix of Q, i.e., $Q = Q_{d+1}$. Denote by N_s the fundamental matrix of Q_s (notation as introduced in Section 5). Let 1_s be the column vector of length $(s+2)$ whose entries are all 1 and $\epsilon_s = N_s 1_s$. For $s = 0, \ldots, d$, ϵ_s is the expected number of rounds to reach state F from state C^{d-s} and ϵ_{d+1} is the expected number of rounds to reach state F from I, i.e., $\epsilon_{d+1} = E[A_d]$ (Theorem 3.3.5, [12]). Identifying P with C^{d+1} we have $\epsilon_{-1} = 1$.

Lemma 15. *The expected number $E[A_d]$ of rounds to reach F from I is at most* 5 *and the variance is at most* 3.6.

Proof. Note that Q_s and N_s are upper triangle matrices. Let

$$E_i - Q_i = \begin{pmatrix} 1-a_1 & -a_2 & \ldots & -a_{i+2} \\ 0 & & & \\ \vdots & & E_{i-1} - Q_{i-1} & \\ 0 & & & \end{pmatrix} \qquad N_i = \begin{pmatrix} x_1 & x_2 & \ldots & x_{i+2} \\ 0 & & & \\ \vdots & & N_{i-1} & \\ 0 & & & \end{pmatrix}$$

$E_i = (E_i - Q_i)N_i$ gives rise to $(i+2)^2$ equations. Adding up the $i+2$ equations for the first row of E_i results in

$$\epsilon_i = (1-a_1)^{-1}\left(1 + \sum_{l=2}^{i+2} a_l \epsilon_{i+1-l}\right) \tag{1}$$

It is straightforward to verify that $\epsilon_{-1} = 1$ and $\epsilon_0 = 3$. Hence

$$\epsilon_i = (1-a_1)^{-1}\left(1 + \sum_{l=2}^{i} a_l \epsilon_{i+1-l} + 3a_{i+1} + a_{i+2}\right)$$

Next we show by induction on i that $\epsilon_i \leq 4$ for $i = -1, 0, 1, \ldots, d$. So assume that $\epsilon_l \leq 4$ for $l = -1, 0, 1, \ldots, i-1$ with $i < d$. Then

$$\epsilon_i \leq (1-a_1)^{-1}\left(1 + 4\sum_{l=2}^{i} a_l + 3a_{i+1} + a_{i+2}\right)$$

since $a_i \geq 0$. Using the fact $1 - a_1 = \sum_{l=2}^{i+2} a_l$ this inequality becomes

$$\epsilon_i \quad \leq \quad (1-a_1)^{-1}(1 + 4(1 - a_1 - a_{i+1} - a_{i+2}) + 3a_{i+1} + a_{i+2}) = 4 + \frac{1 - a_{i+1} - 3a_{i+2}}{1-a_1}$$

Coefficient a_j denotes the transition probability from C^{d-i} to $C^{d+j-(i+1)}$ for $j = 1, \ldots, i+1$ and a_{i+2} that for changing from C^{d-i} to P. For $i \leq d$ the following values from Lemma 14 are used:

$$\begin{aligned} a_1 &= \left(\binom{i}{i+1-l}\left(\frac{1}{2}\right)^{i+1} + \frac{1}{i+1}\binom{i}{l-1}\left(\frac{1}{4}\right)^{i}(3^{i+1-l} - 2^{i+1-l})\right) \\ a_{i+1} &= \left(\frac{1}{2}\right)^{i+1} \text{ and } a_{i+2} = \frac{1}{i+1}\left(\frac{3}{4}\right)^{i} + \frac{i-1}{2(i+1)}. \text{ Thus,} \end{aligned}$$

$$3a_{i+2} = \frac{3}{i+1}\left(\frac{3}{4}\right)^{i} + \frac{3(i-1)}{2(i+1)} > 1$$

holds for $i \geq 2$. This yields

$$\frac{1 - a_{i+1} - 3a_{i+2}}{1-a_1} < 0$$

and therefore $\epsilon_i \leq 4$. To bound ϵ_{d+1} we use Equation 1 with $i = d+1$. Note that in this case $a_1 = 0$ since a transition from I to itself is impossible. Hence

$$E[A_d] = \epsilon_{d+1} = 1 + \sum_{l=2}^{d+3} a_l \epsilon_{d+2-l} \leq 1 + 4\sum_{l=2}^{d+3} a_l = 5$$

Thus, $Var[A_d] = ((2N_{d+1} - E_{d+1})1_{d+1} - 1_{d+1}^2)[1] = 2\sum_{i=1}^{d+3} x_i \epsilon_{d+2-i} - \epsilon_{d+1} - \epsilon_{d+1}^2$ Figure 6c shows that $Var[A_d] \leq 3.6$. □

Lemma 16. *The expected containment time after a memory corruption at node v is at most $\frac{1}{\ln 2} H_{\delta_i(v)} + 11/2$ with variance less than 7.5.*

Proof. For a set X of configurations and a single configuration c denote by $E(c, X)$ the expected value of the number of transitions from x to a state in X. Let $\mathcal{L}$ be the set of legitimate system states. Then

$$\begin{aligned}
E(I,\mathcal{L}) &= \sum_{e\in T(I,\mathcal{L})} l(e)p(e) \\
&= \sum_{x\in F}\sum_{e_1\in T(I,x)}\sum_{e_2\in T(x,\mathcal{L})} (l(e_1)+l(e_2))p(e_1)p(e_2) \\
&= \sum_{x\in F}\sum_{e_1\in T(I,x)}\left(l(e_1)p(e_1)\sum_{e_2\in T(x,\mathcal{L})} p(e_2) + p(e_1)\sum_{e_2\in T(x,\mathcal{L})} l(e_2)p(e_2)\right) \\
&= \sum_{x\in F}\sum_{e_1\in T(I,x)} (l(e_1)p(e_1)+p(e_1)E(x,\mathcal{L})) \\
&= \sum_{x\in F}\left(E(I,x) + \sum_{e_1\in T(I,x)} p(e_1)E(x,\mathcal{L})\right) \\
&\leq E(I,F) + \max\{E(x,\mathcal{L}) \mid x\in F\}\sum_{e_1\in T(I,F)} p(e_1) \\
&= E(I,F) + \max\{E(x,\mathcal{L}) \mid x\in F\} \leq 5 + \frac{1}{\ln 2}H_{\delta_i(v)} + 1/2
\end{aligned}$$

The last step uses Lemma 4 and 15. The bound on the variance is proved similarly. □

Theorem 3 and 4, Lemma 8 and 16 together prove the following Theorem.

Theorem 5. *$\mathcal{A}_{col}$ is a self-stabilizing algorithm for computing a $(\Delta+1)$-coloring in the synchronous model within $O(\log n)$ time with high probability. It uses messages of size $O(\log n)$ and requires $O(\log n)$ storage per node. With respect to memory and message corruption it has contamination radius 1. The expected containment time is at most $\frac{1}{\ln 2}H_{\Delta_i} + 11/2$ with variance less than 7.5.*

Corollary 1. *Algorithm $\mathcal{A}_{col}$ has expected containment time $O(1)$ for bounded-independence graphs. For unit disc graphs this time is at most 8.8.*

Proof. For these graphs $\Delta_i \in O(1)$, in particular $\Delta_i \leq 5$ for unit disc graphs. □

7. Conclusions

The analysis of self-stabilizing algorithms is often confined to the stabilization time starting from an arbitrary configuration. In practice the time to recover from a 1-faulty configuration is much more relevant. This paper presents techniques to analyze the containment time of randomized self-stabilizing algorithms for 1-faulty configurations. The execution of an algorithm is modeled as a Markov chain, its complexity is reduced with the lumping technique. The power of this technique is demonstrated by an application to a $\Delta+1$-coloring algorithm. We believe that the technique can also be applied to other self-stabilizing algorithms. We leave the application to problems such as maximal independents sets and maximal matchings for future work.

Funding: Research was funded by Deutsche Forschungsgemeinschaft DFG (TU 221/6-2).

Conflicts of Interest: The author declares no conflict of interest.

References

1. Gärtner, F.C. Fundamentals of Fault-tolerant Distributed Computing in Asynchronous Environments. *ACM Comput. Surv.* **1999**, *31*, 1–26. [CrossRef]
2. Dolev, S. *Self-Stabilization*; MIT Press: Cambridge, MA, USA, 2000.
3. Azar, Y.; Kutten, S.; Patt-Shamir, B. Distributed Error Confinement. *ACM Trans. Algorithms* **2010**, *6*. [CrossRef]

4. Kutten, S.; Patt-Shamir, B. Adaptive Stabilization of Reactive Protocols. In *Lecture Notes in Computer Science, Proceedings of the International Conference on Foundations of Software Technology and Theoretical Computer Science, Chennai, India, 16–18 December 2004*; Springer: Berlin/Heidelberg, Germany, 2004; Volume 3328, pp. 396–407.
5. Ghosh, S.; He, X. Scalable Self-Stabilization. *J. Parallel Distrib. Comput.* **2002**, *62*, 945–960. [CrossRef]
6. Dubois, S.; Masuzawa, T.; Tixeuil, S. Bounding the Impact of Unbounded Attacks in Stabilization. *IEEE Trans. Parallel Distrib. Syst.* **2012**, *23*, 460–466. [CrossRef]
7. Beauquier, J.; Delaet, S.; Haddad, S. Necessary and sufficient conditions for 1-adaptivity. In Proceedings of the 20th International Parallel and Distributed Processing Symposium, Rhodes Island, Greece, 25–29 April 2006; pp. 10–16.
8. Ghosh, S.; Gupta, A.; Herman, T.; Pemmaraju, S. Fault-containing self-stabilizing distributed protocols. *Distrib. Comput.* **2007**, *20*, 53–73. [CrossRef]
9. Köhler, S.; Turau, V. Fault-containing self-stabilization in asynchronous systems with constant fault-gap. *Distrib. Comput.* **2012**, *25*, 207–224. [CrossRef]
10. Ghosh, S.; Gupta, A. An exercise in fault-containment: Self-stabilizing leader election. *Inf. Process. Lett.* **1996**, *59*, 281–288. [CrossRef]
11. Turau, V.; Hauck, B. A fault-containing self-stabilizing (3 - 2/(Delta+1))-approximation algorithm for vertex cover in anonymous networks. *Theor. Comput. Sci.* **2011**, *412*, 4361–4371. [CrossRef]
12. Kemeny, J.G.; Snell, J.L. *Finite Markov Chains*; Springer: Berlin/Heidelberg, Germany, 1976.
13. Duflot, M.; Fribourg, L.; Picaronny, C. Randomized Finite-state Distributed Algorithms As Markov Chains. In *Lecture Notes in Computer Science, Proceedings of the International Symposium on Distributed Computing, Lisbon, Portugal, 3–5 October 2001*; Springer: Berlin/Heidelberg, Germany, 2001; Volume 2180, pp. 240–254.
14. DeVille, R.; Mitra, S. Stability of Distributed Algorithms in the Face of Incessant Faults. In *Lecture Notes in Computer Science, Proceedings of the Symposium on Self-Stabilizing Systems, Lyon, France, 3–6 November 2009*; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5873, pp. 224–237.
15. Fribourg, L.; Messika, S.; Picaronny, C. Coupling and Self-Stabilization. *Distrib. Comput.* **2006**, *18*, 221–232. [CrossRef]
16. Yamashita, M. Probabilistic Self-Stabilization and Random Walks. In Proceedings of the 2011 Second International Conference on Computing, Networking and Communications (ICNC), Osaka, Japan, 30 November–2 December 2011; pp. 1–7.
17. Mitton, N.; Fleury, E.; Guérin-Lassous, I.; Séricola, B.; Tixeuil, S. On Fast Randomized Colorings in Sensor Networks. In *Proceedings of ICPADS*; IEEE: New York, NY, USA, 2006; pp. 31–38.
18. Crouzen, P.; Hahn, E.; Hermanns, H.; Dhama, A.; Theel, O.; Wimmer, R.; Braitling, B.; Becker, B. Bounded Fairness for Probabilistic Distributed Algorithms. In Proceedings of the 11th International Conference on Application of Concurrency to System Design (ACSD), Newcastle Upon Tyne, UK, 20–24 June 2011; pp. 89–97.
19. Peleg, D. *Distributed Computing: A Locality-Sensitive Approach*; SIAM Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 2000.
20. Gradinariu, M.; Tixeuil, S. Self-stabilizing Vertex Coloring of Arbitrary Graphs. In Proceedings of the 4th International Conference on on Principles of Distributed Systems (OPODIS 2000), Paris, France, 20–22 December 2000; pp. 55–70.
21. Dolev, S.; Herman, T. Superstabilizing Protocols for Dynamic Distributed Systems. *Chic. J. Theor. Comput. Sci.* **1997**, *4*, 1–40.
22. Luby, M. A Simple Parallel Algorithm for the Maximal Independent Set Problem. *SIAM J. Comput.* **1986**, *15*, 1036–1055. [CrossRef]
23. Johansson, Ö. Simple Distributed $\delta + 1$-coloring of Graphs. *Inf. Process. Lett.* **1999**, *70*, 229–232. [CrossRef]
24. Barenboim, L.; Elkin, M. *Distributed Graph Coloring: Fundamentals and Recent Developments*; Morgan & Claypool Publishers: Williston, VT, USA, 2013.
25. Lenzen, C.; Suomela, J.; Wattenhofer, R. Local algorithms: Self-stabilization on speed. In Proceedings of the Symposium on Self-Stabilizing Systems, Lyon, France, 3–6 November 2009; Springer: Berlin/Heidelberg, Germany, 2009; pp. 17–34.