Probabilistic Self-stabilizing Vertex Coloring in Unidirectional Anonymous Networks

Samuel Bernard*, Stéphane Devismes$^\circ$, Katy Paroux†, Maria Potop-Butucaru*, and Sébastien Tixeuil*

* Université Pierre et Marie Curie - Paris 6
$^\circ$ Université Grenoble I
† INRIA Bretagne Atlantique

Abstract. A distributed algorithm is self-stabilizing if after faults and attacks hit the system and place it in some arbitrary global state, the system recovers from this catastrophic situation without external intervention in finite time. Unidirectional networks preclude many common techniques in self-stabilization from being used, such as preserving local predicates. The focus of this work is on the classical vertex coloring problem, that is a basic building block for many resource allocation problems arising in wireless sensor networks.
In this paper, we investigate the gain in complexity that can be obtained through randomization. We present a probabilistically self-stabilizing algorithm that uses `k` states per process, where `k` is a parameter of the algorithm. When $\mathtt{k} = \Delta + 1$, the algorithm recovers in expected $O(\Delta n)$ actions. When `k` may grow arbitrarily, the algorithm recovers in expected $O(n)$ actions in total. Thus, our algorithm can be made optimal with respect to space or time complexity. Our case study hints that randomization could help filling the complexity gap between bidirectionnal and unidirectionnal networks.

Keywords: Wireless sensor networks, distributed algorithms, self-stabilization, unidirectional anonymous networks, lower and upper bounds, coloring problem, randomization.

1 Introduction

Wireless sensor networks are already used in a variety of fields, like home appliances, irrigation, medicine, monitoring, real-time control systems, military defense applications, etc. Recent advances in hardware design and manufacturing, computing and storage capabilities of the sensing devices themselves, have made it practically possible to envision very large size sensor networks comprising hundreds of thousands or even millions of autonomous sensor nodes. The scalability requirements of future wireless sensor networks drives new problems to application designers. For example, it becomes unrealistic to ensure human maintainance of such networks, to assume unique identifiers will be available, or to expect every sensor node to have exactly the same communication capabilities as its neighbors. Moreover, when sensor networks are deployed in unattended environments, it is likely that faults and attacks will hit the system.

One of the most versatile technique to ensure forward recovery of distributed systems is that of *self-stabilization* [6, 7]. A distributed algorithm is self-stabilizing if after faults and attacks hit the system and place it in some arbitrary global state, the system recovers from this catastrophic situation without external (*e.g.* human) intervention in finite time. Self-stabilization makes no hypotheses about the extent or the nature of the faults and attacks that may harm the system, yet may induce some overhead (*e.g.* memory, time) when there are no faults, compared to a classical (*i.e.* non-stabilizing) solution. Computing space and time bounds for particular problems in a self-stabilizing setting is thus crucial to evaluate the impact of adding forward recovery properties to the system.

The vast majority of self-stabilizing solutions in the literature [7] considers *bidirectional* communications capabilities, *i.e.* if a process u is able to send information to another process v, then v is always able to send information back to u. This assumption is valid in many cases, but cannot capture the fact that asymmetric situations may occur, *e.g.* in wireless sensor networks, it is possible that u is able to send information to v yet v cannot send any information back to u (u may have a wider range antenna than v). Asymmetric situations, that we denote in the following under the term of *unidirectional* networks, preclude many common techniques in self-stabilization from being used, such as preserving local predicates (a process u may take an action that violates a predicate involving its outgoing neighbors without u knowing it, since u cannot get any input from them).

Related works. Investigating the possibility of self-stabilization in unidirectional networks such as those resulting from wireless communication medium was recently emphasized in several papers [1, 3–5, 8–10][1]. In particular, [4] shows that in the simple case of acyclic unidirectional networks, nearly any recursive function can be computed anonymously in a self-stabilizing way. Computing global tasks in a general topology requires either unique identifiers [1, 3, 8] or distinguished processes [5, 9, 10].

The paper most related to our work [2] studies *deterministic* solutions to the self-stabilizing vertex coloring problem in *unidirectional networks*. To satisfy the vertex coloring specification in unidirectional networks, an algorithm must ensure that no two neighboring nodes (*i.e.* two nodes u and v such that either u can send information to v, or v can send information to u, but not necessarily both) have identical colors. When deterministic solutions are considered, [2] proves a lower bound of n states per process (where n is the network size) and a recovery time of at least $n(n-1)/2$ actions in total (and thus $\Omega(n)$ actions per process). [2] also presents a deterministic algorithm for vertex coloring with matching upper bounds that performs in arbitrary graphs.

[1] We do consider here the overwhelming number of contributions that assume a unidirectional ring shaped network, please refer to [7] for additional references

Those high lower bounds results contrast with the many low upper bounds existing in the litterature about *bidirectional* networks. Indeed, both deterministic and probabilistic solutions to the vertex coloring problem [11, 13] in bidirectional networks require only a number of states that is proportional to the network maximum degree Δ, and the number of actions per process in order to recover is $O(\Delta)$ (in the case of a deterministic algorithm) or expected $O(1)$ (in the case of a probabilistic one). Moreover, since the length of the chain of causality after a correcting action is executed is only one, strict Byzantine containement can be achieved [14].

Our contribution. In this paper, we investigate the possibility of lowering complexity results for the vertex coloring problem in *unidirectional networks* by means of randomization. We first observe that at least $\Delta+1$ states per process and a recovery time of $\Omega(n)$ actions in total (and thus $\Omega(1)$ actions per process) are required. We present a probabilistically self-stabilizing algorithm for vertex coloring that uses `k` states per process, where `k` is a parameter of the algorithm. When $k = \Delta+1$ (*i.e.* when the number of used colors is optimal), the algorithm recovers in expected $O(\Delta n)$ actions in total. When `k` may grow arbitrarily, the algorithm recovers in expected $O(n)$ actions in total (*i.e.* an optimal – constant – number of actions per node). Thus, our algorithm can be made optimal with respect to space or time complexity. This results solves the open question of [2] with respect to the computing power of probabilistic protocols. Our results are particularly well suited to dynamic wireless sensor networks as we make no hypothesis about the availability of unique identifiers for the various nodes, *i.e.* the participants are completely *anonymous*.

Outline. The remaining of the paper is organized as follows: Section 2 presents the programming model and problem specification, while Section 3 presents our randomized solution to the problem. Section 4 gives some concluding remarks and open questions.

2 Model

Distributed Program model. A distributed program consists of a set V of n processes which may not have unique identifiers. Therefore, processes will be referred in the following as anonymous. A process maintains a set of variables that it can read or update, that define its *state*. Each variable ranges over a fixed domain of values. We use small case letters to denote singleton variables, and capital ones to denote sets. A process contains a set of *constants* that it can read but not update. A binary relation E is defined over distinct processes such that $(i,j) \in E$ if and only if j can read the variables maintained by i; i is a *predecessor* of j, and j is a *successor* of i. The set of predecessors (resp. successors) of i is denoted by $P.i$ (resp. $S.i$), and the union of predecessors and successors of i is denoted by $N.i$, the *neighbors* of i. In some case, we are interested in the iterated notions of those sets, *e.g.* $S.i^0 = i$, $S.i^1 = S.i$, $\ldots$, $S.i^k = \cup_{j \in S.i} S.j^{k-1}$.

The values $\delta_{in}.i$, $\delta_{out}.i$, and $\delta.i$ denote respectively $|P.i|$, $|S.i|$, and $|N.i|$; Δ_{in}, Δ_{out}, and Δ denote the maximum possible values of $\delta_{in}.i$, $\delta_{out}.i$, and $\delta.i$ over all processes in V.

An action has the form $\langle name \rangle : \langle guard \rangle \longrightarrow \langle command \rangle$. A *guard* is a Boolean predicate over the variables of the process and its communication neighbors. A *command* is a sequence of statements assigning new values to the variables of the process. We refer to a variable v and an action a of process i as $v.i$ and $a.i$ respectively. A *parameter* is used to define a set of actions as one parameterized action.

A *configuration* of the distributed program is the assignment of a value to every variable of each process from its corresponding domain. Each process contains a set of actions referred in the following as *algorithm*. In the following we consider that processes are *uniform*. That is, all the processes contain the exact same set of actions. An action is *enabled* in some configuration if its guard is **true** in this configuration. A *computation* is a maximal sequence of configurations such that for each configuration γ_i, the next configuration γ_{i+1} is obtained by executing the command of at least one action that is enabled in γ_i. Maximality of a computation means that the computation is infinite or it eventually reaches in a terminal configuration where none of the actions are enabled. *silent*.

A *scheduler* is a predicate on computations, that is, a scheduler is a set of possible computations, such that every computation in this set satisfies the scheduler predicate. The *unfair distributed* scheduler, that we use in the sequel, corresponds to predicate **true** (that is, all computations are allowed).

A configuration *conforms* to a predicate if this predicate is **true** in this configuration; otherwise the configuration *violates* the predicate. By this definition every configuration conforms to predicate **true** and none conforms to **false**. Let R and S be predicates over the configurations of the program. Predicate R is *closed* with respect to the program actions if every configuration of the computation that starts in a configuration conforming to R also conforms to R. Predicate R *converges* to S if R and S are closed and any computation starting from a configuration conforming to R contains a configuration conforming to S. The program *deterministically stabilizes* to R if and only if **true** converges to R. The program *probabilistically stabilizes* to R if and only if **true** converges to R with probability 1.

Problem specification. Consider a set of colors ranging from 0 to $\mathtt{k}-1$, for some integer $\mathtt{k} \geq 1$. Each process i defines a function *color.i* that takes as input the states of i and its predecessors, and outputs a value in $\{0, \ldots, \mathtt{k}-1\}$. The *unidirectional vertex coloring* predicate is satisfied if and only if for every $(i, j) \in E$, $color.i \neq color.j$.

3 Probabilistic self-stabilizing unidirectional coloring

We first observe two lower bounds that hold for any kind of program that is self-stabilizing or probabilistically self-stabilizing for the unidirectional coloring specification:

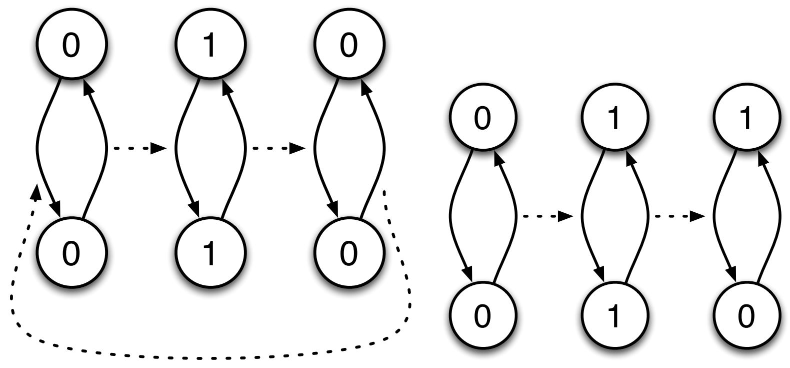
1. *The minimal number of states per process is $\Delta + 1$.* Consider a bidirectional clique network (that is $(\Delta+1)$-sized), and assume the output of the coloring protocol is now fixed (that is, the network is vertex colored). Suppose that only Δ states per process are used, then at least two processes i and j have the same state, and have the same view of their predecessors. As a result $color.i = color.j$, and i and j being neighbors, the unidirectional coloring predicate does not hold in this terminal configuration.
2. *The minimal number of moves overall is $\Omega(n)$.* Consider a unidirectional chain of processes which are all initially in the same state. For every process but one, the color is identical to that of its predecessor. Since a change of state may only resolve two conflicts (that of the moving node and that of its successor), a number of overall moves at least equal to $\lfloor n/2 \rfloor$ is required, thus $\Omega(n)$ moves.

The algorithm presented as Algorithm 1 can be informally described as follows. If a process has the same color as one of its predecessors then it chooses a new color in the set of available colors (*i.e.* the set of colors that are not already used by any of its predecessors), or retains its current color.

The algorithm is only slightly different from its classical self-stabilizing *bidirectional* counterpart [11] by the fact that a node may retain its own color (the random color is always changed in [11]). This is due to the fact that some particular networks could drive the classical probabilistic protocol into a deterministic behavior. The example presented in Figure 1.*a* depicts an execution of a two nodes network such that boths nodes are both predecessor and successor of one another. With our definition, each node has $\delta.i = 1$, and thus only two colors are available for each of them. If node may not reuse their own color, then they must choose a different one. Only one such color is available, and unfortunately this color is the same for both of them. As a result, when both nodes start with the same color 0, and are always scheduled for execution simultaneously, they both choose the same (different form 0) color 1. The argument can then be repeated to induce an infinite loop that never stabilizes. Such a scenario may not happen with our scheme, since there always exists a positive probability that even if selected simultaneously, two neighboring such node choose different colors (see execution in Figure 1.*b*).

The colors are chosen in a set of size `k`, where `k` is a parameter of the algorithm, using uniform probability $1/\mathtt{k}$. In the following, we show that Algorithm 1 is probabilistically self-stabilizing for the unidirectional coloring problem if $\mathtt{k} > \Delta$. To reach that goal we proceed in two steps: first we show that any terminal configuration satisfies the unidirectional coloring predicate (Lemma 1); secondly, we show that the expected number of steps to reach a terminal configuration starting from an arbitrary one is bounded (Lemma 6).

Lemma 1. *Any terminal configuration satisfies the unidirectional coloring predicate.*



(a) Degenerate deterministic execution (b) Still probabilistic execution

Fig. 1. Executions of coloring process in degenerate networks

Algorithm 1 A uniform probabilistic coloring algorithm for general unidirectional networks

```
process $i$
const
    k : integer
    $P.i$ : set of predecessors of $i$
    $C.i$ : set of colors of nodes in $P.i$
parameter
    $p$ : node in $P.i$
var
    $c.i$ : color of node $i$
function
    random($S$ : set of colors) : color
        // returns a color in $S$ chosen with uniform probability
action
    $p \in P.i$, $c.i = c.p \rightarrow$
        $c.i :=$ random $((\{0, \ldots, \mathtt{k} - 1\} \setminus C.i) \cup \{c.i\})$
```

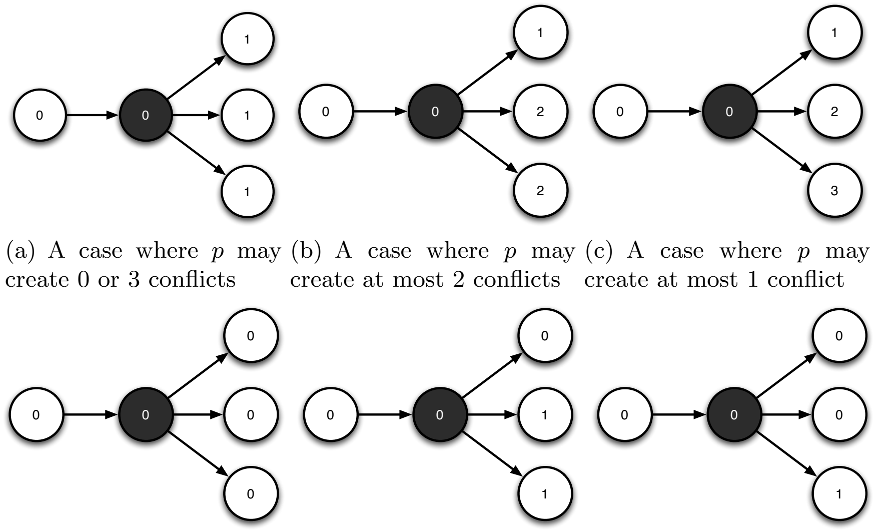
Proof. In a terminal configuration, every process i satisfies $\forall j \in P.i, c.i \neq c.j$ and $\forall j \in S.i, c.i \neq c.j$. Hence, in a terminal configuration, every process i has a color that is different from those of its neighbors, which proves the lemma.

Definition 1 (Conflict). *Let p be a process and γ a configuration. The tuple (p,γ) is called a* conflict *if and only if there exists $q \in P.p$ (the predecessors of p) such that $c.q = c.p$ in γ.*

Lemma 2. *Assume $\mathtt{k} > \Delta$. The number X_1 of conflicts created by the color change of Process p is equal to a sum of $\alpha.p$ Bernoulli random variables with common parameter $\beta.p$, where $\alpha.p = \delta.p - \delta_{in}.p$ and*

$$\beta.p = \frac{1}{\mathtt{k} - |C.p|} \leqslant \frac{1}{\mathtt{k} - \delta_{in}.p}. \tag{1}$$

Proof. The idea of the proof is illustrated in Figure 2 for $\mathtt{k} = 5$, $\delta.p = 4$, and $\delta_{in}.p = 1 = |C.p|$, where process p is denoted in dark gray. The random variable X_1 is the sum of 3 Bernoulli random variables [12] with common parameter 1/4.



(a) A case where p may create 0 or 3 conflicts (b) A case where p may create at most 2 conflicts (c) A case where p may create at most 1 conflict

(d) A case where p may not create any conflict (e) A case where p may create 0 or 2 conflicts (f) A case where p may create at most 1 conflict

Fig. 2. Illustrating the number of conflicts with color change, for $\mathtt{k} = 5$, $\delta.p = 4$, and $\delta_{in}.p = 1$

When a process p actually changes its color, this new color is chosen in a set of $\mathtt{k} - |C.p|$ colors which contains at least $\mathtt{k} - \delta_{in}.p$ colors. That is, there are $\mathtt{k}$ colors and it cannot choose a color chosen by one of its predecessors, therefore

$|C.p|$ colors are removed from the set of possible choices, which means that at most $\delta_{in}.p$ colors are removed.

Let q be a successor of p which is not a predecessor of p. After p changes its color, p and q are in conflict if and only if p chooses the color of q. The probability that p chooses the color of q is equal to one over the number of choices for the color of p: $1/(\mathtt{k} - |C.p|)$.

Since the number of successors of p not in the set of predecessors of p is $\delta.p - \delta_{in}.p$, the number of conflicts created by p is a sum of $\alpha.p$ Bernoulli random variables taking value one with probability $1/(\mathtt{k} - |C.p|)$, where $\alpha.p = \delta.p - \delta_{in}.p$. Since these Bernoulli random variables are in general not independent, one cannot expect their sum to follow the binomial law. However, their dependence does *not* impact the expectation of their sum.

Lemma 3. *Assume $\mathtt{k} > \Delta$. The probability of the event CCp that the Process p does change its color is equal to:*

$$\mathbb{P}(CCp) = \frac{\mathtt{k} - |C.p|}{\mathtt{k} - |C.p| + 1}. \tag{2}$$

As a consequence, the number Y_1 of conflicts (same or new) induced by the activation of Process p admits the following bound for its expectation:

$$\mathbb{E}(Y_1) \leqslant \frac{\delta.p - \delta_{in}.p + 1}{\mathtt{k} - \delta_{in}.p + 1}. \tag{3}$$

Proof. Since all colors have the same probability to appear, the probability that Process p changes its color is the number $\mathtt{k} - |C.p|$ of choices for change over the total number of choices $\mathtt{k} - |C.p| + 1$. Thus the probability of this event is:

$$\mathbb{P}(CCp) = \frac{\mathtt{k} - |C.p|}{\mathtt{k} - |C.p| + 1}. \tag{4}$$

The number of conflicts induced by the activation of Process p is equal to the number of new conflicts created if it changes its color and to one (still itself) if it keeps its color. Using conditonnal expectation [16] (which corresponds in such discrete case to standard expectation with respect to a conditionnal probability), we have that

$$\mathbb{E}(Y_1) = \mathbb{E}(Y_1|CCp) \times \mathbb{P}(CCp) + \mathbb{E}(Y_1|CCp^c) \times \mathbb{P}(CCp^c), \tag{5}$$

where CCp^c denotes the complementary of the event CCp. Thus we obtain

$$\mathbb{E}(Y_1) = \mathbb{E}(X_1) \times \mathbb{P}(CCp) + 1 \times \mathbb{P}(CCp^c), \tag{6}$$

where X_1 has the same meaning as in Lemma 2, and thanks to lemma 2 we obtain

$$\mathbb{E}(Y_1) = \frac{\delta.p - \delta_{in}.p + 1}{\mathtt{k} - |C.p| + 1} \leqslant \frac{\delta.p - \delta_{in}.p + 1}{\mathtt{k} - \delta_{in}.p + 1}. \tag{7}$$

Lemma 4. *Assume* $\mathtt{k} > \Delta$. *The expected number of conflicts induced by an activation of Process* p *is less than or equal to:*

$$\mathtt{M} = \frac{\Delta}{\mathtt{k}} < 1. \tag{8}$$

Proof. Observe that for all $p \in V, \Delta \geqslant \delta.p$, therefore for all $p \in V$,

$$\frac{\delta.p - \delta_{in}.p + 1}{\mathtt{k} - \delta_{in}.p + 1} \leqslant \frac{\Delta - \delta_{in}.p + 1}{\mathtt{k} - \delta_{in}.p + 1}. \tag{9}$$

In order to find an upper bound for this value, let the function f be

$$f : x \in [1, \Delta] \mapsto \frac{\Delta - x + 1}{\mathtt{k} - x + 1}. \tag{10}$$

Its derivative exists and is equal to

$$f'(x) = \frac{\Delta - \mathtt{k}}{(\mathtt{k} - x + 1)^2}. \tag{11}$$

By hypothesis, we have that $\mathtt{k} > \Delta$, so $f'(x) < 0$ for all $x \in [1, \Delta]$ and f is decreasing. Therefore, $f(x)$ is maximal when $x = 1$, which leads to

$$\frac{\delta.p - \delta_{in}.p + 1}{\mathtt{k} - \delta_{in}.p + 1} \leqslant \frac{\Delta - \delta_{in}.p + 1}{\mathtt{k} - \delta_{in}.p + 1} \leqslant \frac{\Delta}{\mathtt{k}}. \tag{12}$$

Lemma 5. *Assume* $\mathtt{k} > \Delta$. *Let* (p, γ) *be a conflict. The expected number of steps necessary to solve this conflict is less than:*

$$\frac{1}{1 - M} = \frac{\mathtt{k}}{\mathtt{k} - \Delta}. \tag{13}$$

Proof. The expected number of conflicts created by a single process is given by Lemma 4. Let us explain how we deal with the second stage when resolving conflicts, then any further stage follows by induction. Our idea is to introduce the following probabilistic tool. We consider a branching tree which is a generalisation of a Galton-Watson tree [15], where the number of offsprings are dependant random variables. As previously mentioned, this dependency does not matter when computing the expectation.

Recall that Y_1 is the number of conflicts induced by an activation of a Process p. We will denote by Y_2 the number of conflicts created at the second stage. As observed above,

– either process p changes its color and then the number Y_2 of conflicts induced by the activation of the associated Y_1 Processes is equal to a sum of Y_1 random variables T_q where q is a successor of p not in the set of predecessors of p and in conflict with p;
– or process p keeps its color, and $Y_1 = 1$, the number Y_2 of conflicts created will be equal in law to Y_1.

In the first case, each T_q has the same behaviour as $Y_1 = T_p$ (with different parameters) and is independant from Y_1. Then we have the following:

$$\mathbb{E}(Y_2) \leqslant \mathbb{E}\left(\sum_{j=1}^{Y_1} T_{q_j}\right) = \sum_{y=1}^{\alpha.p}\left(\mathbb{E}(\sum_{j=1}^{y} T_{q_j})\,\mathbb{P}(Y_1 = y)\right)$$
$$\leqslant \sum_{y=1}^{\alpha.p}\left(y[\max_q \mathbb{E}(T_q)]\mathbb{P}(Y_1 = y)\right) = \mathbb{E}(Y_1)\,M \leqslant M^2.$$

By induction, the expected number of conflicts $\mathbb{E}(Y_\ell)$ created at stage ℓ is less than

$$\mathbb{E}(Y_1)\,M^{\ell-1} \leqslant M^\ell.$$

Since $M < 1$, we have a convergent geometric series. Thus the expected total number of conflicts is less than $1/(1-M)$.

Lemma 6. *Assume* $\mathtt{k} > \Delta$. *Starting from an arbitrary configuration, the expected number of color changes to reach a configuration verifying the unidirectional coloring predicate is less than or equal to:*

$$\frac{n\,\mathtt{k}}{\mathtt{k}-\Delta}. \tag{14}$$

Proof. In the worst case the number of initial conflicts is n. Then the proof is a direct consequence of Lemma 5.

Theorem 1. *Algorithm 1 is a probabilistic self-stabilizing solution for the unidirectional coloring when* $\mathtt{k} > \Delta$.

Proof. The proof is a direct consequence of Lemmas 1 and 6.

Notice that with a minimal number of colors (*i.e.*, $\mathtt{k} = \Delta + 1$), the expected number of steps to reach a terminal configuration starting from an arbitrary configuration is less than $n(\Delta+1)$. Moreover, when the number of colors increases (*i.e.*, $\mathtt{k} \to \infty$), the expected number of steps to reach a terminal configuration starting from an arbitrary configuration converges to n.

4 Conclusion

We investigated the intrinsic complexity of performing local tasks in unidirectional anonymous networks in a self-stabilizing setting. Contrary to "classical" bidirectional networks, local vertex coloring now induces global complexity (n states per process at least, n moves per process at least) for *deterministic* solutions. By contrast, we presented asymptotically optimal solutions for the *probabilistic* case (that actually match the bounds obtained for bidirectional networks). This work raises several important open questions:

1. Our probabilistic solution can be tuned to be optimal in space (and is then with a Δ multiplicative penalty in time), or optimal in time, but not both. However, our lower bounds do not preclude the existence of probabilistic solutions that are optimal for both complexity measures.
2. Our lower bound on the number of colors is generic (it should hold for graphs of any shape), while the *chromatic number* of a graph denote the actual number of colors that are needed to color a particular graph. It is worth investigating whether our protocol can color particular graphs with a lower number of colors.

References

1. Yehuda Afek and Anat Bremler-Barr. Self-stabilizing unidirectional network algorithms by power supply. *Chicago J. Theor. Comput. Sci.*, 1998, 1998.
2. Samuel Bernard, Stéphane Devismes, Maria Gradinariu Potop-Butucaru, and Sébastien Tixeuil. Optimal deterministic self-stabilizing vertex coloring in unidirectional anonymous networks. In *Proceedings of the IEEE International Conference on Parallel and Distributed Processing Systems (IPDPS 2009)*, pages 1–8, Rome, Italy, May 2009. IEEE Press.
3. Jorge Arturo Cobb and Mohamed G. Gouda. Stabilization of routing in directed networks. In Ajoy Kumar Datta and Ted Herman, editors, *WSS*, volume 2194 of *Lecture Notes in Computer Science*, pages 51–66. Springer, 2001.
4. Sajal Das, Ajoy Kumar Datta, and Sébastien Tixeuil. Self-stabilizing algorithms in dag structured networks. *Parallel Processing Letters (PPL)*, 9(4):563–574, December 1999.
5. Sylvie Delaët, Bertrand Ducourthial, and Sébastien Tixeuil. Self-stabilization with r-operators revisited. *Journal of Aerospace Computing, Information, and Communication (JACIC)*, 3(10):498–514, 2006.
6. Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974.
7. Shlomi. Dolev. *Self-stabilization.* MIT Press, March 2000.
8. Shlomi Dolev and Elad Schiller. Self-stabilizing group communication in directed networks. *Acta Inf.*, 40(9):609–636, 2004.
9. Bertrand Ducourthial and Sébastien Tixeuil. Self-stabilization with r-operators. *Distributed Computing (DC)*, 14(3):147–162, July 2001.
10. Bertrand Ducourthial and Sébastien Tixeuil. Self-stabilization with path algebra. *Theoretical Computer Science (TCS)*, 293(1):219–236, February 2003.
11. Maria Gradinariu and Sébastien Tixeuil. Self-stabilizing vertex coloring of arbitrary graphs. In *Proceedings of International Conference on Principles of Distributed Systems (OPODIS 2000)*, pages 55–70, Paris, France, December 2000.
12. Anders Hald. *A history of probability and statistics and their applications before 1750.* Wiley Series in Probability and Mathematical Statistics: Probability and Mathematical Statistics. John Wiley & Sons Inc., New York, 1990. A Wiley-Interscience Publication.
13. Nathalie Mitton, Eric Fleury, Isabelle Guérin-Lassous, Bruno Séricola, and Sébastien Tixeuil. On fast randomized colorings in sensor networks. In *Proceedings of ICPADS 2006*, pages 31–38. IEEE Press, July 2006.

14. Mikhail Nesterenko and Anish Arora. Tolerance to unbounded byzantine faults. In *21st Symposium on Reliable Distributed Systems (SRDS 2002)*, pages 22–29. IEEE Computer Society, 2002.
15. J. R. Norris. *Markov chains*, volume 2 of *Cambridge Series in Statistical and Probabilistic Mathematics*. Cambridge University Press, Cambridge, 1998. Reprint of 1997 original.
16. Sheldon M. Ross. *Introduction to probability models*. Harcourt/Academic Press, Burlington, MA, seventh edition, 2000.