Brief Announcement: Memory Lower Bounds for Self-Stabilization

Lélia Blin
Sorbonne Université, CNRS, Université Evry-Val d'Essonne, LIP6 UMR 7606, Paris
lelia.blin@lip6.fr

Laurent Feuilloley
Sorbonne Université, CNRS, LIP6 UMR 7606, Paris
laurent.feuilloley@lip6.fr

Gabriel Le Bouder
Sorbonne Université, CNRS, ENS Paris-Saclay, LIP6 UMR 7606, Paris
gleboude@ens-paris-saclay.fr

Abstract

In the context of self-stabilization, a *silent* algorithm guarantees that the communication registers (a.k.a register) of every node do not change once the algorithm has stabilized. At the end of the 90's, Dolev et al. [Acta Inf. '99] showed that, for finding the centers of a graph, for electing a leader, or for constructing a spanning tree, every silent deterministic algorithm must use a memory of $\Omega(\log n)$ bits per register in n-node networks. Similarly, Korman et al. [Dist. Comp. '07] proved, using the notion of proof-labeling-scheme, that, for constructing a minimum-weight spanning tree (MST), every silent algorithm must use a memory of $\Omega(\log^2 n)$ bits per register. It follows that requiring the algorithm to be silent has a cost in terms of memory space, while, in the context of self-stabilization, where every node constantly checks the states of its neighbors, the silence property can be of limited practical interest. In fact, it is known that relaxing this requirement results in algorithms with smaller space-complexity.

In this paper, we are aiming at measuring how much gain in terms of memory can be expected by using arbitrary deterministic self-stabilizing algorithms, not necessarily silent. To our knowledge, the only known lower bound on the memory requirement for deterministic general algorithms, also established at the end of the 90's, is due to Beauquier et al. [PODC '99] who proved that registers of constant size are not sufficient for leader election algorithms. We improve this result by establishing the lower bound $\Omega(\log \log n)$ bits per register for deterministic self-stabilizing algorithms solving $(\Delta + 1)$-coloring, leader election or constructing a spanning tree in networks of maximum degree Δ.

2012 ACM Subject Classification Theory of computation → Distributed computing models

Keywords and phrases Space lower bound, memory tight bound, deterministic self-stabilization, leader election, anonymous, identifiers, state model, ring

Digital Object Identifier 10.4230/LIPIcs.DISC.2019.37

Related Version http://arxiv.org/abs/1905.08563

Funding Support by ANR ESTATE.

1 Introduction

Self-stabilization is a suitable paradigm for asynchronous distributed systems subject to transient failures. The occurrence of failures can bring the system into arbitrary configurations. A self-stabilizing algorithm guarantees a return to a correct behavior in finite time, without external intervention. The legality of the configuration is a notion that depends on the problem considered. For instance, for the problem of $(\Delta + 1)$-coloration of the nodes, a configuration is legal if every node has a color in $[1, \ldots, \Delta + 1]$ different from the color of each of its neighbors.

In the state model, the processes have two types of memory. The *immutable* memory is used to store the identity of the node, its ports numbers, and the code of the algorithm. By definition the immutable memory is fault free. On the other hand, the *mutable* memory is the memory used to store variables, this memory is not fault free. In fact, each node has only one register and the register of each node is composed by the mutable memory. Recall that in this model, each node uses its register to communicate the values of its own variables to its neighbors. As a result, only mutable memory (a.k.a register) is considered for computing *memory complexity* a.k.a *space complexity* of a self-stabilizing protocol, and immutable memory is not. Note that a node can use to compute the information of its immutable memory, such as its identity.

Indeed, small space-complexity is desirable for several reasons. One reason is for reducing the overhead due to *link congestion* [1] (note that the nodes carry on exchanging information, even after stabilization, and even when no fault occurs). Another reason is that mixing *variable replication* with self-stabilization can be desirable [8], but replication is possible only if the overall memory occupied by these variables is small.

In this paper, we establish a lower bound of $\Omega(\log\log n)$ bits space memory for the register of each node for the leader election problem, the $(\Delta+1)$-coloration of the nodes (denoted by *coloring* problem), and the spanning tree construction. Note that we consider a network where each node $v \in V$ has a distinct identity, denoted by $ID(v) \in \{1,\ldots,n^c\}$ for some constant $c > 1$. This significantly improves the only lower bound [2] known so far, from $\Omega(1)$ to $\Omega(\log\log n)$. Moreover, our lower bound invalidates the folklore conjecture stating that the aforementioned problems might be solvable using only $O(\log^* n)$ bits by register. More importantly, our lower bound implies that the upper bound $O(\log\Delta+\log\log n)$ bits of register per node in [6] for the *coloring* problem and spanning tree construction is space optimal.

In the context of self-stabilization, a *silent* algorithm guarantees that the register of every node does not change once the algorithm has stabilized. The memory efficiency in the context of silent self-stabilizing algorithms is well studied. Firstly, Dolev and al. [7], at the end of the 90's, proved that finding the centers of a graph, leader election, and spanning tree construction require a memory of $\Omega(\log n)$ bits per register whenever the algorithm is requested to be silent. On the other hand, the design of silent algorithms is based on a mechanism known as *proof-labeling-scheme* (PLS) [10], and space lower bounds for PLSs imply space lower bounds for silent self-stabilizing algorithms. A typical example is the $\Omega(\log^2 n)$-bit lower bound on the size of every PLS for minimum-weight spanning trees (MST) [9] , which implies the same bound for constructing an MST in a silent manner [4]. It follows that requiring the algorithm to be silent has a cost in terms of memory space, while, in the context of self-stabilization, where every node constantly checks the states of its neighbors, the silence property can be of no practical interest. In fact, it is known that relaxing this requirement results in algorithms with smaller space-complexity [5, 6].

2 Lower bounds

2.1 Main theorem

▶ **Theorem 1.** *Let $c > 1$. Every self-stabilizing deterministic algorithm under the fair distributed scheduler solving $(\Delta+1)$-coloring, leader election, or spanning tree construction in n-node graphs where every node has a unique identities in $[1, n^c]$ requires $\Omega(\log\log n)$ bits of memory per register at each node.*

The proof is to be found in the full version [3]. The core of the proof is the following. It is known that problems such as coloring, leader election and spanning tree construction require identities to break symmetry under a distributed scheduler. In other words, no algorithm

can solve such problems in anonymous networks. In essence, we show that an algorithm in a network with identities but in which the size of each register is too small does not have more power than an algorithm running in an anonymous network. More specifically, let A be an algorithm in a network with identities, and let us assume that A uses $o(\log\log n)$ bits by register. Observe that even if the network has identities, and even if these identities are used by the algorithm, their size is not taken into account in the space complexity of the algorithm, as identities are not stored in register, but in the immutable memory. However, the nodes cannot write their identities in the register, which is too small, and the identities have to be transferred between nodes in a series of smaller pieces of information. We show that, with only $o(\log\log n)$ bits node register, there exist graphs and identity assignments to the nodes of these graphs such that the algorithm A has the same behavior as an algorithm in the anonymous version of these graphs. It follows that A cannot solve coloring problem, leader election and spanning tree construction.

Observe that it is usually assumed that any coloring algorithm must compute a color variable $c_v \in [1, \ldots, \Delta + 1]$ at each node v. It follows that such coloring algorithms require registers of size $\Omega(\log \Delta)$ bits. Similarly, it is usually assumed that any algorithm constructing a spanning tree computes a port number $p_v \in [1, \ldots, \Delta]$ at each node $v \in V$. Hence, such algorithms require registers of size $\Omega(\log \Delta)$ bits. Consequently, the deterministic self-stabilizing algorithms for coloring and for spanning tree construction in [6] are space-optimal. Indeed, these algorithms use $O(\log \Delta + \log\log n)$ bits of memory at each node.

References

1. J. Adamek, M. Nesterenko, and S. Tixeuil. Evaluating Practical Tolerance Properties of Stabilizing Programs through Simulation: The Case of Propagation of Information with Feedback. In *SSS 2012*, pages 126–132, 2012.
2. J. Beauquier, M. Gradinariu, and C. Johnen. Memory Space Requirements for Self-Stabilizing Leader Election Protocols. In *PODC 1999*, pages 199–207, 1999.
3. L. Blin, L. Feuilloley, and G. Le Bouder. Memory Lower Bounds for Self-Stabilization. *CoRR*, abs/1905.08563, 2019. `arXiv:1905.08563`.
4. L. Blin and P. Fraigniaud. Space-Optimal Time-Efficient Silent Self-Stabilizing Constructions of Constrained Spanning Trees. In *ICDCS 2015*, pages 589–598, 2015.
5. L. Blin and S. Tixeuil. Compact deterministic self-stabilizing leader election on a ring: the exponential advantage of being talkative. *Distributed Computing*, 31(2):139–166, 2018.
6. L. Blin and S. Tixeuil. Compact Self-Stabilizing Leader Election for General Networks. In *LATIN 2018*, pages 161–173, 2018.
7. S. Dolev, M. G. Gouda, and M. Schneider. Memory Requirements for Silent Stabilization. *Acta Inf.*, 36(6):447–462, 1999.
8. T. Herman and S. V. Pemmaraju. Error-detecting codes and fault-containing self-stabilization. *Inf. Process. Lett.*, 73(1-2):41–46, 2000.
9. A. Korman and S. Kutten. Distributed verification of minimum spanning trees. *Distributed Computing*, 20(4):253–266, 2007.
10. A. Korman, S. Kutten, and D. Peleg. Proof labeling schemes. *Distributed Computing*, 22(4):215–233, 2010.