Report Number: WUCSE-2012-62

2012

Studying Network Optimization in the Context of Self-Stabilization

Authors: Abusayeed Saifullah

Self-stabilization is a theoretical framework of non-masking fault-tolerance for distributed networks. A self-stabilizing system is capable of tolerating any unexpected transient fault without outside intervention and, regardless of the initial state, it can converge to a legitimate global state, a predefined vector of local states, in finite time. Self-stabilization has rendered a good problem solving paradigm of networks over the last decade. In this paper, we survey the self-stabilizing solutions for various network optimization problems such as network flow, load balancing, load and resource distribution, routing, file distribution, shortest paths etc. The paper also summarizes some recent works presenting how the convergence of a self-stabilizing distributed network can be modelled as a convex optimization problem with the exploitation of an analogy between self-stabilizing systems and stable feedback systems. The works pertaining to gradient adaptation of self-stabilizing system are also presented.
... **Read complete abstract on page 2.**

Studying Network Optimization in the Context of Self-Stabilization

Complete Abstract:

Self-stabilization is a theoretical framework of non-masking fault-tolerance for distributed networks. A self-stabilizing system is capable of tolerating any unexpected transient fault without outside intervention and, regardless of the initial state, it can converge to a legitimate global state, a predefined vector of local states, in finite time. Self-stabilization has rendered a good problem solving paradigm of networks over the last decade. In this paper, we survey the self-stabilizing solutions for various network optimization problems such as network flow, load balancing, load and resource distribution, routing, file distribution, shortest paths etc. The paper also summarizes some recent works presenting how the convergence of a self-stabilizing distributed network can be modelled as a convex optimization problem with the exploitation of an analogy between self-stabilizing systems and stable feedback systems. The works pertaining to gradient adaptation of self-stabilizing system are also presented.

STUDYING NETWORK OPTIMIZATION IN THE CONTEXT OF SELF-STABILIZATION

Abusayeed Saifullah
Department of Computer Science and Engineering
Washington University in St Louis
saifullaha@cse.wustl.edu

ABSTRACT

Self-stabilization is a theoretical framework of non-masking fault-tolerance for distributed networks. A *self-stabilizing system* is capable of tolerating any unexpected transient fault without outside intervention and, regardless of the initial state, it can converge to a *legitimate global state*, a predefined vector of local states, in finite time. Self-stabilization has rendered a good problem solving paradigm of networks over the last decade. In this paper, we survey the self-stabilizing solutions for various network optimization problems such as network flow, load balancing, load and resource distribution, routing, file distribution, shortest paths etc. The paper also summarizes some recent works presenting how the convergence of a self-stabilizing distributed network can be modelled as a convex optimization problem with the exploitation of an analogy between self-stabilizing systems and stable feedback systems. The works pertaining to gradient adaptation of self-stabilizing system are also presented.

1 Introduction

The notion of *self-stabilization* was introduced by Dijkstra in 1974 in his classic paper [47]. Dijkstra [47] studies the scenario when there is no global memory and, therefore, the current global state of the system is recorded in variables distributed over the various processes where the communication facilities are restricted only between neighbors. Based on the network topology and signal propagation delay, each node gets only a partial view of the global state. Dijkstra [47] notices the complication that the behavior of a process can only be influenced by that part of the total current system state description that is available to it; local actions taken on account of local information must accomplish a global objective.

Uncertain as to whether the local moves can assure convergence towards satisfaction of such a global criterion, Dijkstra [47] limited his attention to a ring of finite-state machines and provided its solution for self-stabilization which he proved later in [48]. This was the idea that did break the ice to encompass a formal and unified approach to fault tolerance. For almost one decade following the inception, this brilliant work was not noticed and very few papers were published in this area. Once this proposal was recognized as a milestone in works on fault tolerance, the notion propagated among the distributed system community rapidly that resulted in a burst of papers [128, 125, 126, 127, 129] as well as workshops and symposiums devoted entirely

to this area.

The adoption of self-stabilization is of great importance where a system is vulnerable to transient faults. In modern distributed networks, various transient faults are likely to occur as these systems are exposed to constant changes of their environment. Since most self-stabilizing algorithms are non-terminating, if the system experiences transient faults corrupting the local states of the computing elements, once faults cease, the algorithms themselves guarantee to recover in finite time to a safe state without being assisted by any outside agent. This also means that the complicated task of initializing the system is no longer needed, since the algorithms regain correct behavior regardless of initial state. Self-stabilization provides a formal and unified approach to fault tolerance with respect to a model of transient failures and makes the departure from previous approaches to fault tolerance in distributed network.

At present, self-stabilization has rendered a good problem solving paradigm for distributed networks. Many network optimization problems such as network flow, load balancing, routing, load and resource distribution, file distribution, shortest paths etc. have been solved in the context of self-stabilization. In this paper, we survey the self-stabilizing solutions of network optimization problems available in the literature (Section 3 to Section 13). Some recent works presenting how the convergence of a self-stabilizing distributed network can be modelled as a convex optimization problem with the exploitation of an analogy between self-stabilizing systems and stable feedback systems are also presented (Section 3). We also summarize the works pertaining to gradient adaptation of self-stabilizing systems (Section 6). Finally, we we propose some related promising open problems along with some concluding remarks in Section 14.

2 Preliminaries

Some concepts and definitions related to this survey are in order and are explained as follows:

2.1 Distributed Network

A ***distributed system*** is generally defined as a set of processing elements or state machines interconnected by a network of some fixed topology. The types of hardware, operating systems, and other resources may vary drastically. It is similar to computer clustering with the main difference being a wide geographic dispersion of the resources. In the shared register or local shared memory model of distributed network, communication is restricted only between neighbors. That is, there is no common or global memory. Each processor can communicate only with the neighbors and the communication is carried out by using a shared communication channel or link. Directly connected processors are called each other's *neighbors*. By contrast, in a message passing model of distributed system, communications are carried out by sending and receiving messages.

From the local and global perspective two types of states are defined:

Local State: Each node maintains a set of local variables whose contents specify the *local state* of that node.

Global State: This is the state of the system as a whole. The *global state* is expressed in terms of the local states of the processors. Specifically, the *global state* is a vector $c \in (s_1 \times s_2 \times \cdots \times s_n)$, where s_i is the local state of machine i.

2.2 Fault Tolerance

Implicit in the definition of *fault tolerance* is the assumption that there is a specification of what constitutes the correct state and behavior of the system. A *failure* occurs when an actual running system deviates from this specification [130]. The cause of a failure is called an *error*. An error represents an invalid system state, one that is not allowed by the system behavior specification. The error itself is the result of a defect in the system or fault. In other words, a fault is the root cause of a failure [130].

Most of the phenomena that contribute to the unexpected perturbation of the system state are indistinguishable. Some of them, explained in [130] and [129], are:

- **Processing Machine Failures**
- **Inconsistent Initialization**
- **Mode Change**
- **Communication Media Failures**
- **Transmission Delays**
- **Distributed Agreement Problems**

Based on duration, faults can be classified as *transient* or *permanent* [130]. A *transient fault* is an event that may change the state of a system by corrupting the local states of the machines. A particularly problematic type of transient fault is the intermittent fault that recurs, often unpredictably [130].

Fault tolerance is the ability of a system to perform its function correctly even in the presence of internal faults. Fault-tolerant computing is extremely hard since it involves intricate algorithms to cope with the inherent complexity of the physical world. As it turns out that the world conspires against us and is constructed in such a way that, generally, it is simply not possible to devise an absolutely foolproof or hundred percent reliable system [60]. No matter how hard we try, there is always a possibility that something can go wrong. The best we can do is to reduce the probability of failure to an acceptable level. Unfortunately, the more we strive to reduce this probability, the higher becomes the cost.

2.3 Self-Stabilizing System

The property of self-stabilization can recover the system from transient faults and represents a departure from previous approaches to fault tolerance. The global correctness criterion of the system is defined by a *predicate* which is a boolean function over the whole system. Based on this predefined function, two classes of global states are defined:

Legitimate State: When the system satisfies the predicate it is said that the system is in a *legitimate state*.

Illegitimate State: The state of the system is said *illegitimate* when it fails to satisfy the predicate.

The goal of a self-stabilizing algorithm is to start from an arbitrary (possibly illegitimate) state and then to

reach a legitimate state after a finite number of steps (moves). For each machine one or more privileges are defined. A *privilege* (also known as *guard*) of a processor is a boolean function of its own state and the states of its neighbors. A processor that satisfies its privilege can make a move. A *move* is an action taken by a processor that changes the local state of the processor. Therefore, each move takes the system into a new global state.

In addition to this predicate of global correctness criterion, Dijkstra [47] defines the following four constraints of a legitimate state of the system:

1. In each legitimate state one or more privileges will be present.
2. In each legitimate state, each possible move will bring the system again in another legitimate state.
3. Every privilege exists in at least one legitimate state.
4. For any pair of legitimate states, there exists a sequence of moves that can transfer the system from one state to the other.

However, depending on the system specifications and criteria of the problems, these requirements have been modified in some papers. Schneider [129] introduces a generalization of self-stabilization based on Arora and Gouda [11] and Arora [10]. He defines a system X as self-stabilizing with respect to a predicate P if X satisfies the following two properties:

1. **Closure:** P is closed under the execution of X. That is, once P is established in X, it cannot be falsified.
2. **Convergence:** Starting from an arbitrary global state, X is guaranteed to reach a global state satisfying P within a finite number of state transitions (moves).

A self-stabilizing algorithm can be encoded as a set of rules. Each rule has two parts: the *privilege* and the *move* as shown below:

$$\textbf{if} < privilege > \textbf{then} < move >$$

If two or more processors satisfy their privileges at the same time, only one of them is arbitrarily selected by a *daemon* (central or distributed). Two variations of self-stabilization, called *super-stabilization* and *silent stabilization*, are proposed in [55] and [50], respectively.

3 Convergence Analysis

Since verification of the convergence (to predicate or correct global state) criterion is tedious, the vision is to provide for proofs of self-stabilization in an automatic fashion. One excellent but difficult strategy to prove the correctness is to use a bounded function defined over the entire system. The bounded function has been used in [62, 90, 35, 91, 93] whose value monotonically decreases with computing steps. However, in [6, 134, 8], no bounded function has been used. Correctness can also be proven by mathematical induction [8, 9] and graph theoretical reasoning [134, 6].

Oehlerking et al. [116] study the convergence to the global correctness criterion from a very different perspective. The paper points out the analogy between self-stabilizing systems and *stable feedback systems* and thus self-stabilization and *Lyapunov stabilization* are treated exactly alike. A continuous-time linear

time-invariant system S is said to be *Lyapunov stable* if and only if all the *eigenvalues* of S have real parts less than or equal to 0, and those with real parts equal to 0 are nonrepeated. When the system behaves like a *piecewise-affine hybrid system*, i.e.

$x[k+1] = A_m x + b_m$, $A_m \in \Re^{n\times n}$, $b_m \in \Re^n$, where x is the vector of states of all machines of the system, then the convergence to the predicate is simply the well-known *convex optimization* [27]. Let $x[k+1] = f(x[k])$ (since the state change is the function of previous state) be a discrete-time system with $f(\underline{0}) = \underline{0}$. If $\exists V : \Re^n \to \Re$ such that $V(x)$ is positive definite, $\dot{V}(x) = V(f(x)) - V(x)$ is negative definite, and $V(x) \to \infty$ for $\|x\| \to \infty$, then V is a Lyapunov function and the system is globally asymptotically stable in $\underline{0}$. Oehlerking et al. [116] use quadratic Lyapunov function candidates and reduce the problem of finding a Lyapunov function to *convex optimization* via so-called *linear matrix inequalities* problem:

Find $k_i \in \Re$ such that $x^T(F_0 + \sum_{i=1}^{N} k_i F_i)x \geq 0$, $\forall x \in \Re^n$, where $F_j \in \Re^{n\times n}, 0 \leq j \leq N$, are fixed.

Herman [85] and Beauquier et al. [22] explain convergence from the probabilistic point of view. In [85], for token ring circulation, the convergence properties are interpreted in terms of a function D that partitions the set of ring states with at least two tokens, where $D.x$ is the minimum distance between two tokens in ring state x. The expected value of k such that $D.(f^k.x) = 1$ is:

$$E[kC.k] = \sum_{i=0}^{\infty} i e_j A^i {e_1}^T = e_j(\sum_{i=0}^{\infty} i A^i){e_1}^T \quad (1)$$

where vector e_j represents states satisfying $D.x = j$, vector $e_j A$ contains the probabilities for $D.(f.x)$, vector $e_j A^k$ contains the probabilities for $D.(f^k.x)$, and f is the function that takes a ring state as input and outputs a ring state. The closed form of (1) is obtained by transform analysis:

$$E[kC.k] = e_j(I-A)^{-1}A(I-A)^{-1}{e_1}^T \quad (2)$$

The expected time for convergence follows from (2):

$E[1 + kC.k/4] = 1 + E[kC.k]/4 = n^2/2$.

Kakugawa [102] introduces a description language and its processor for self-stabilizing systems of arbitrary network topology to verify mechanically. For an incorrect self-stabilizing system, the verification system outputs a counterexample which consists of an initial system state and an execution sequence which does not converge.

4 Time and Space Optimality

Great efforts have been given to achieve time, space, and state optimality [28] in the adoption of self-stabilization towards solving different problems in recent years. Aggarwal [3] is the first self-stabilizing algorithm to compute the spanning tree of an asynchronous network which is time-optimal (i.e. stabilizes in time O($diameter$)) without any prior knowledge of the network size. It presents both a randomized Las-Vegas algorithm for anonymous network and a deterministic version of ID-based network. In [49], time optimal self-stabilizing dynamic protocols for a variety of tasks including routing, leader election, topology update are available. Each of this protocols sta-

bilizes in O(d) time, where d is the diameter of the system.

Gradinariu and Johnen [81] propose a self-stabilizing probabilistic solution for the neighborhood unique naming problem in uniform, anonymous networks with arbitrary topology. Their solution is time optimal i.e. naming is done in only one trial per processor in the average under any unfair scheduler. Efficiency and simplicity for self-stabilizing token circulation in trees and arbitrary networks have been extensively studied by Petit [120]. For different network topology the service time and space of this protocol have been analyzed in [95, 22, 96, 19]. Space optimality for stabilizing leader election is achieved in the randomized algorithm of Beauquir et al. [20]. Awerbuch [15] considers the question of fault-tolerant distributed network protocols with extremely small memory requirements per processor. Specifically, he shows that even in the case of worst-case transient faults, many fundamental network protocols can be achieved using only O($\log n$) bits of memory per incident network edge. A time-optimal self-stabilizing network synchronization algorithm is provided in [14]. Ghodsi [67] proposes a self-stabilizing network size estimation gossip algorithm for peer to peer network. Memory requirements for several silent stabilizing algorithms have been analyzed by Dolev [50]. Dolev [50] shows that any center-finding, leader election or tree construction algorithm of this class require $\Omega(\log n)$ bits per communication register or process.

5 Optimized Network Flow

A combination of the optimized network flow algorithms of Ford-Fulkerson [63] and Edmonds-Karp [59] has been implemented in the self-stabilizing model by Ghosh et al. [72]. The algorithm uses an approach similar to that of Goldberg and Tarjan [79]. For each edge (i, j), there is a register or variable $f(i, j)$ at both node i and node j. At every node i, the variable $d(i)$ contains the believed shortest path from s (source) to i. For every node $i \neq s, t$, $demand(i) = O_f(i) - I_f(i)$, where $O_f(i)$ and $I_f(i)$, at i, are outflow and inflow, respectively (and t is the sink). Each node i, tries to restore flow conservation constraint, $demand(i) = 0$, either by reducing inflow or increasing outflow if $demand(i) < 0$, or by increasing inflow or reducing outflow if $demand(i) > 0$. Each node with positive demand attempts to pull flow via a shortest path from s to itself in the residual graph. If the node believes that a path from s to itself does not exist in the residual graph, then it rejects the demand by pushing it back along an outgoing edge.

Four types of privileges or guarded commands are[1]:

$S_1 : d(i) \neq min(\{d(p) + 1 | p \in IN(i)\} \cup \{n\}) \Rightarrow d(i) := min(\{d(p) + 1 | p \in IN(i)\} \cup \{n\})$;

$S_2 : demand(i) < 0 \Rightarrow Reduce\_Inflow(i)$;

$S_3 : (\exists j \in IN(i) : demand(i) > 0 \wedge d(i) < n \wedge d(j) = d(i) - 1) \Rightarrow (f(j, i) := f(j, i) + min(demand(i), r(j, i)))$;

$S_4 : demand(i) > 0 \wedge d(i) = n \wedge i \neq t \Rightarrow Reduce\_Outflow(i)$;

[1] $IN(i)$: incoming edges of i; $C(i, j)$: capacity of (i, j); $r(i, j) = C(i, j) - f(i, j)$;

An illustration of the algorithm is shown in Figure 1 and Table 1.

In the legitimate state, $d(i)$ is the shortest path from s to i and f-values of the edges constitute a maximum flow. The convergence requires O(n^2) moves.

Move	Node	Privilege
$A \rightarrow B$	a	S_2
$B \rightarrow C$	b	S_1
$C \rightarrow D$	t	S_3
$D \rightarrow E$	b	S_3
$E \rightarrow F$	a	S_3
$F \rightarrow G$	b	S_1
$G \rightarrow H$	b	S_4
$H \rightarrow I$	t	S_1

Table 1: Moves and corresponding nodes and guards of Figure 1

In [80], a formal definition of routing metrics and two necessary and sufficient conditions to maximize a routing metric over a tree have been presented. The paper generalizes the maximum flow tree which it calls the *maximal metric tree* that can stabilize in O(n^2) time.

6 Gradient Algorithm

The self-stabilizing gradient approach proposed by Doughlas [57] mimics in spirit the recently developed self-stabilized subspace methods [58, 117]. Douglas [57] extends the previous ideas in the development of self-stabilization for gradient adaptation of orthonormal matrices to develop algorithms for instantaneous prewhitened blind separation of homogeneous signal mixtures. In the problem of *blind source separation* (BSS), an n-dimensional vector sequence $x(k)$ is assumed to be produced from an n-dimensional source signal vector as:

$$x(k) = Ax(k) \tag{3}$$

where A is an unknown nonsingular $n \times n$ mixing matrix and the elements of $s(k)$ are zero-mean, unity variance, symmetrically distributed, and statistically independent of each other. The goal of the BSS task is to process each $x(k)$ via a linear transformation M such that

$$MA = \Phi D \tag{4}$$

where Φ and D are $(n \times n)$ permutation and diagonal scaling matrices, respectively. The output signal vector, $y(k) = Mx(k)$, contains scaled versions of all of the elements of $s(k)$ without crosstalk. It can be assumed without loss of generality that $E[s(k)s^T(k)] = I$, such that $R_{xx} = E[x(k)x^T(k)]$, and let P be a prewhitening matrix such that

$$PR_{xx}P^T = PAA^TP^T = \Gamma\Gamma^T = I \tag{5}$$

where $\Gamma = PA$ is an orthonormal matrix. By defining $W(k)P = M$, the optimum solution of $W(k)$ can be shown to be of the form:

$$W_{opt} = \Phi J \Gamma^T \tag{6}$$

where J is an orthogonal matrix of ± 1's.

Cardoso et al. [32] define the prewhitened BSS problem as:

$$\text{Maximize } \beta \sum_{i=1}^{n} E[\phi_i(y_i(k))] \tag{7}$$

$$\text{such that } W(k)W^T(k) = I. \tag{8}$$

Figure 1: An illustration of maximum flow algorithm (d-values are shown below each node)

where $\beta < 0$, $\phi_i(y_i) = -\log p_i(y_i)$, and $p_i(y_i)$ is the assumed density model for the source signal extracted at the i-th system output. This problem is equivalent to minimizing the Kullback Leibler divergence measure given by [31]: $D(p_s \| \hat{p}_s) = \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} p_s(s) \log \left(\frac{p_s(s)}{\hat{p}_s(s)} \right) ds_1 \cdots ds_n$;

under the constraint in (8), where $p_s(s)$ is the source vector probability density function.

The algorithms in [57] are approximate gradient procedures in the inhomogeneous space of all $n \times n$ orthonormal matrices satisfying constraint (8), a matrix space also known as the Stiefel manifold [84]. These algorithms are self-stabilizing in the sense that the projection of back to the space of orthonormal matrices is not required i.e. the rows of the adaptive demixing matrix do not need to be periodically reorthonormalized. All small perturbations of the demixing matrices away from orthonormality do not accumulate over time. The algorithms contribute to the growing body of work on gradient algorithms within constrained parameter spaces.

7 TDMA Slot Assignment

Kulkarni et al. [111] is the first to develop self-stabilizing TDMA. The paper starts from the view of a grid topology and assumes that each nodes knows its location in the grid and thus generates a TDMA schedule. But it requires the grid mapping be same for all networks and the locations be known before the TDMA algorithm starts. Hence this algorithm is not adoptable in dynamic systems. In the self-stabilizing setting, the most studied vertex coloring

problem is $L(1,0)$. Ghosh and Karaata [73] provide an elegant algorithm for coloring planar graphs, Sur and Srimani [135] provide the same algorithm for bipartite graphs, Shukla et al. [132] provide algorithms for complete odd-degree bipartite graphs and tree graphs, while a solution for general graph is given by Gradinariu and Tixeuil [82]. Recently, Gradinariu and Johnen [81] have presented a solution to $L(1,1)$ coloring , using a number of colors proportional to n^2, where n is the number of nodes in the system. A common drawback of these algorithms lies behind the strong assumption of the existence of a reliable and powerful communication system. The algorithm by Herman [86] is tailored to networks with sensor nodes and computes a TDMA schedule in a distributed manner. Thereby an upper bound on the number of nodes in a neighborhood is assumed. The algorithm consists of five parts running concurrently in an endless loop. Shared variables are propagated to the neighbors of a node by message exchange in a globally synchronized CSMA/CA slot. The whole process is expected to converge locally in constant time and globally in sub linear time.

8 Routing Algorithms

The notion of self-stabilization has recently been adopted into routing algorithms. In [44], a hierarchical routing algorithm from self-stabilizing point of view is provided. Johnen et al. [100] report the first self-stabilizing Border Gateway Protocol (BGP), the standard inter-domain routing protocol in the Internet. The protocol is combined with an existing protocol to make it resilient to policy conflicts. A self-stabilizing loop-free routing algorithm is reported in [101] that is also route preserving. Unlike the previous approaches, it does not require that a bound on the network diameter is known to the processors that perform the routing algorithm. The self-stabilizing cluster routing algorithm in [24] for MANET is based on link-cluster architecture. The algorithm selects the clusterheads, and then builds, in those nodes, routing tables regarding nodes inside and outside the cluster. Shen et al. [131] give a self-stabilizing routing protocol for publish-subscribe system. Neighboring message routers periodically exchange their routing table state, and take corrective actions if (and only when) necessary. Datta et al. [42] present the first self-stabilizing network algorithm in the wormhole routing model, using the unidirectional ring topology. The solution benefits from wormhole routing by providing high throughput and low latency, and front self-stabilization by ensuring automatic resilience to all possible transient failures.

9 Load and Resource Distribution

Gärtner et al. [66] propose a self-stabilizing algorithm for equal load distribution among the replicated servers. Server load is measured in the number of accesses it receives within a certain period of time. The vector ${S_i}^t$ is the state of node i at real time t. A phase θ_i of node i is a fixed time interval of t_i. A node performs a regular access pattern if and only if the accesses of the node to other nodes (and thus local state transitions) occur at the same instances in time within every phase, i.e. ${S_i}^t = {S_i}^{t+\theta_i}$. The client side of the distribution module (between client and server) receives special server instruction messages which it

uses to guide the distribution process of subsequent requests by the application. The data structures of a client consist of two vectors: T-vector, C-vector. The contents of T describe the access pattern of the client within the last phase while C contains the accumulated access statistics from the current phase. The algorithm uses the access pattern T from the last phase to select a server for the next incoming request from the application. Each replicated server maintains a C-vector of current access statistics from the clients. From time to time the server receives a message from the load balancing module indicating that the load is currently high and that a certain amount of load (namely periodic accesses) should be moved to another server. The system converges to the following predicate and, at this state, loads (i.e. client accesses) get uniformly distributed among the servers:

$$\forall i \in clients : i.C^t = i.C^{t-\theta} \wedge i.T^t = i.T^{t-\theta} \wedge$$

$$\forall j \in servers : j.C^t = j.C^{t-\theta}$$

The self-stabilizing load balancing algorithm of Kam et al. [105] considers a ring of processors and ensures that a task will be scheduled on every functioning processor. On the other hand, Flatebo et al. [61] assumes that the load is completely determined by the tasks that are scheduled and do not depend on other processing that is being done. This algorithm works for an arbitrary network of processors and attempts to distribute tasks optimally over the system.

Ko et al. [109] present a decentralized, fully distributed, scalable protocol that places replicated resources in a network of arbitrary topology such that the furthest distance one must travel to find a particular copy of a resource is only slightly larger than optimal, and the distance between identical copies is large.

10 Global Optimization

For maximal matching, self-stabilizing algorithms are constructed in [89, 88] and are shown to run in linear time by Hedetniemi et al. [83]. A generalization is given in [75]. For maximum matching in trees, Ghosh et al. [68] introduce two-phase methodology: forming a rooted directed tree from an undirected tree in the first phase and then stabilization in a bottom-up manner in the second phase. These algorithms work for anonymous networks. Recently, for large ID-based networks, Goddard et al. [76, 77] have proposed a variation of maximal matching, called *strong matching*, a matching M with the added constraint that no two edges in M are joined by an edge.

Goddard et al. [77] also propose a minimal total dominating set and its generalization which is similar to [74]. A minimal dominating set must be maintained to optimize the number and locations of resource centers in a network. Dominating sets with specific properties have been used in several routing proposals for mobile networks. A synchronous self-stabilizing version of minimal domination protocol is given by Xu et al. [137]. A similar problem, called a *maximal k-packing*, for large networks, has been solved by Goddard et al. [77] in self-stabilizing fashion. This generalizes 2-packing of Karaata [106] and Gairing et al. [64].

In [52], a uniform dynamic self-stabilizing leader election algorithm is available. Self-stabilizing leader election has also been proposed in [20, 19, 7, 52, 90, 70] for different network topology. Self-stabilizing

methods for building minimum spanning trees in symmetric network graph and arbitrary network are available in [4] and [8], respectively. A minimal spanning tree must be maintained to minimize latency and bandwidth requirements of multicast or broadcast messages or to implement echo-based distributed algorithms [77]. Ghosh and Gupta [71] present a technique to transform dynamic programming into self-stabilizing distributed model that runs on trees for designing algorithms for optimization problems. For a tree with radius r this transformation stabilizes in O(r) rounds [52].

11 Distributed File System

Dolev and Kat [53] have given the first self-stabilizing distributed file system. The system constructs and maintains a spanning tree consisting of the servers that have volume replica and caches for the specific file volume. The design is based on a self-stabilizing maintenance of a distributed replica tree for each volume (the update algorithm); the tree provides a communication and consistency layer. File system updates use the tree to implement file read and write operations.

12 Shortest Path Problems

A self-stabilizing solution for the shortest path problem in a distributed network is provided by Huang and Lin [93]. For every node i, the distance from source r to i is maintained in a local variable $d(i)$ of i. The weight of an edge (i, j) is denoted by $w(i, j)$ while $N(i)$ denotes the neighbors of i. The algorithm converges to the following predicate:

$\forall i \neq r, d(i) = min_{j \in N(i)}(d(j) + w(i, j))$;

A variation of this problem, called *all-pairs shortest path*, has been solved in self-stabilizing fashion by Chandrasekar and Srimani [33]. Self-stabilizing shortest path trees (distance-vector) can also be constructed from the maximal metric trees constructed by the self-stabilizing algorithm of Gouda et al. [80].

13 Exploration, Communication, and Token Circulation in the Network

Efficient self-stabilizing algorithms for searching over the network both in breadth-first manner [91, 134] and in depth-first manner [37, 38] resulting in a spanning tree of the network are available in the literature. However, different self-stabilizing algorithms for constructing spanning tree (not necessarily minimum spanning tree) are found in [35, 6, 65, 3]. The spanning tree algorithm in [17] works in wireless ad hoc network. Chaudhuri [34] proposes a self-stabilizing algorithm for minimum depth-search in a network.

Bein and Datta [23] design a self-stabilizing communication protocol in a sensor network, based on the directed diffusion method. A request for data from an initiator node is broadcast in the network, and the positive answers from the sensors are forwarded back to the initiator following a Shortest-Path-Tree construction rooted at the initiator. The protocol proposed by Awerbuch et al. [16] is the first self-stabilizing protocol for end to end communication. Its message complexity is comparable with the corresponding non-stabilizing solutions. Howell et al. [87] define a finite state message passing model which is particu-

larly appropriate for defining and reasoning about self-stabilizing protocols, due to the well known result that self-stabilizing protocols on unbounded-channel models must have infinitely many legitimate states. Dolev and Schiller [54, 56] present the first randomized algorithm for implementing self-stabilizing group communication services in an asynchronous system. Mizuno et al. [114] present two lock based self-stabilizing distributed mutual exclusion algorithms: one is a link-locking algorithm and the other is a node-locking algorithm. The quorum-based mutual exclusion algorithm in [115] scales well since it has constant synchronization delay and its message complexity is O($\sqrt{n}$).

Johnen et al. [99] provide a space-efficient depth-first token circulation algorithm on a uniform rooted network. The single token circulation algorithm of Herman [85] converges in $n^2/2$ expected time. This protocol works for ring of identical machines. Different self-stabilizing token circulation algorithms have also been proposed in [120, 121, 43, 99, 123] for different network topology.

14 Conclusion

Self-stabilization has emerged as a promising paradigm for the design, maintenance, and analysis of fault tolerant distributed networks. As shown in [57], the self-stabilizing modifications also may prove useful for other problems in optimization and numerical linear algebra. This may also be adopted for congestion control, multi-commodity flow, and optimal routing (in future). All types of resource and load distribution may also be solved in this model.

One criticism of self-stabilization as a design goal is that it is too strong a property and thus either too difficult to achieve or can be achieved at the expense of other goals. The complexity analysis is complicated and becomes worse than the corresponding non-self-stabilizing algorithm. Under some strong assumptions, Abello et al. [1] show that any computational problem can be realized in a self-stabilizing fashion. On the other hand, Schneider [129] shows that there are some systems that are not, at all, compatible for self-stabilization. A system may experience some global state from where it cannot recover itself. This type of global state is called an *unsafe* state. It is pointed out in [129] that if any unsafe global state is a final state, then the system will not be able to stabilize.

References

[1] ABELLO, J., AND DOLEV, S. On the computational power of self-stabilizing systems. *Theoretical Computer Science 182*, 1–2 (1997), 159–170.

[2] AFEK, Y., KUTTEN, S., AND YUNG, M. The local detection paradigm for self-stabilizing algorithm. *In Proceedings of the international workshop on distributed computing (Bari, Italy)* (1990).

[3] AGGARWAL, S., AND KUTTEN, S. Time optimal self-stabilizing spanning tree algorithm. In *Proceedings of the 13th Conference on Foundations of Software Technology and Theoretical Computer Science(FSTTCS93), Springer-Verlag LNCS:761* (1993), pp. 400–410.

[4] ANTONOIU, G., AND SRIMANI. A self-stabilizing distributed algorithm for minimal spanning tree in a symmetric graph. *Computers and Mathematics with Applications 35*, 10 (May 1998), 15–23.

[5] ANTONOIU, G., AND SRIMANI, P. Mutual exclusion between neighboring nodes in a tree that stabilizes using read/write atomicity. In *European Conference on Parallel Processing* (September 1998), pp. 545–553.

[6] ANTONOIU, G., AND SRIMANI, P. K. A self-stabilizing distributed algorithm to construct an arbitrary spanning tree of a connected graph. *Computers and Mathematics with Applications 30*, 9 (November 1995), 1–7.

[7] ANTONOIU, G., AND SRIMANI, P. K. A self-stabilizing leader election algorithm for tree graphs. *Journal of Parallel and Distributed Computing 34*, 2 (May 1996), 227–232.

[8] ANTONOIU, G., AND SRIMANI, P. K. Distributed self-stabilizing algorithm for minimum spanning tree construction. *European Conference on Parallel processing* (1997), 480–487.

[9] ANTONOIU, G., AND SRIMANI, P. K. A self-stabilizing distributed algorithm to find the median of a tree graph. *Journal of Computer and System Science 58*, 1 (February 1999), 215–221.

[10] ARORA, A. A foundation for fault-tolerant computing. *PhD dissertation, Univ. of Texas at Austin* (1992).

[11] ARORA, A., AND GOUDA, M. G. Closure and convergence: A foundation for fault-tolerant computing. *In Proceedings of the 22nd International Conference on the Fault-tolerant Computing Systems* (1992).

[12] ARORA, A., AND NESTERENKO, M. Unifying stabilization and termination in message-passing systems. *Distributed Computing 17*, 3 (2005), 279–290.

[13] AWERBUCH, B. Complexity of network synchronization. *J. ACM 32* (1985), 804–823.

[14] AWERBUCH, B., KUTTEN, S., MANSOUR, Y., AND SHAMIR, B. P. Time optimal self-stabilizing synchronization. *In Proceedings of the 25th Symposium on Theory of Computing* (1993).

[15] AWERBUCH, B., AND OSTROVSKY, R. Memory-efficient and self-stabilizing network reset. *In Proc. of the 13th ACM Symp. on Principles of Distributed Computing* (August 1994), 254–263.

[16] AWERBUCH, B., SHAMIR, B. P., AND VARGHESE, G. Self-stabilizing end to end communication. *In Proceedings of the 32nd IEEE symposium on Foundations of Computer Scienceinternational workshop on distributed computing (Bari, Italy)* (October 1991).

[17] BAALA, H., FLAUZAC, O., GABER, J., BUI, M., AND EL-GHAZAWI, T. A. A self-stabilizing distributed algorithm for spanning tree construction in wireless ad hoc networks. *Journal of Parallel and Distributed Computing 63*, 1 (January 2003), 97–104.

[18] BEAUQUIER, J., AND DELAT, S. Probabilistic self-stabilizing mutual exclusion in uniform rings. In *Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing, Los Angeles, California, United States* (1994), SIGACT: ACM Special Interest Group on Algorithms and Computation Theory SIGOPS: ACM Special Interest Group on Operating Systems, ACM Press New York, NY, USA, p. 378.

[19] BEAUQUIER, J., GRADINARIU, M., AND JOHNEN, C. Memory space requirements for self-stabilizing leader election protocols. In *Symposium on Principles of Distributed Computing* (1999), pp. 199–207.

[20] BEAUQUIER, J., GRADINARIU, M., AND JOHNEN, C. Randomized self-stabilizing and space optimal leader election under arbitrary scheduler on rings. Tech. Rep. 99-1225, Université Paris Sud, 1999.

[21] BEAUQUIER, J., GRADINARIU, M., AND JOHNEN, C. Token-based self-stabilizing uniform algorithms. *Journal of Parallel and Distributed Computing 62*, 5 (May 2002), 899–921.

[22] BEAUQUIER, J., AND JOHNEN, C. Analyze of randomized self-stabilized algorithms under non-deterministic scheduler classes. Tech. Rep. LRI, Université Paris Sud, 1999.

[23] BEIN, D., AND DATTA, A. K. A self-stabilizing directed diffusion protocol for sensor networks. *International Conference on Parallel Processing* (August 2004), 69–76.

[24] BEIN, D., DATTA, A. K., JAGGANAGAR, C. R., AND VILLAIN, V. A self-stabilizing link-cluster algorithm in mobile ad hoc networks. *Proceedings of the 8th International Symposium on Parallel Architectures,Algorithms and Networks* (2005), 436–441.

[25] BELKOUCH, F., BUI, M., AND CHENG, L. Self-stabilizing deterministic network decomposition. *Journal of Parallel and Distributed Computing 62*, 4 (April 2002), 696–714.

[26] BLAIR, J. R. S., AND MANNE, F. Efficient self-stabilizing algorithms for tree networks. In *Proceedings. 23rd International Conference on Distributed Computing Systems* (19-22 May 2003), IEEE, pp. 20–26.

[27] BOYD, S., AND VANDEBERGHE, L. *Convex Optimization*.

[28] BUI, A., DATTA, A., PETIT, F., AND VILLAIN, V. State-optimal snap-stabilizing pif in tree networks. *In Proceedings of the Fourth Workshop on Self-Stabilizing Systems (Austin, Texas, USA)* (June 1999), 78–85.

[29] BURNS, J. E., GOUDA, M., AND MILLER, R. On relaxing interleaving assumptions. *In Proceedings of the MCC Workshop on Self-stabilizing Systems, MCC technical Report No. STP-379-89* (1989).

[30] BUSKENS, R. W., AND BIANCHINI, R. P. Self-stabilizing mutual exclusion in the presence of faulty nodes. *In Proceedings of the 25th International Symposium on Fault Tolerant Computing Digest of Papers* (1995), 144–153.

[31] CARDOSO, J.-F. Infomax and maximum likelihood in source separation. *IEEE Signal Processing Lett. 4* (April 1997), 112–114.

[32] CARDOSO, J.-F., AND LAHELD, B. Equivariant adaptive source separation. *IEEE Trans. Signal Processing 44* (December 1996), 3017–3030.

[33] CHANDRASEKAR, S., AND SRIMANI, P. K. A self-stabilizing distributed algorithm for all-pairs shortest path problem. *Parallel Algorithms and Applications 4* (1994), 125–137.

[34] CHAUDHURI, P. A self-stabilizing algorithm for minimum-depth search of graphs. *Information Processing Letters 118*, 1-4 (September 1999), 241–249.

[35] CHEN, N.-S., YU, H.-P., AND HUANG, S.-T. A self-stabilizing algorithm for constructing spanning trees. *Information Processing Letters 39*, 3 (August 1991), 147–151.

[36] COLLIN, Z., DECHTER, R., AND KATZ, S. Self-stabilizing distributed constraint satisfaction. *Chicago Journal of Theoretical Computer Science* (1999).

[37] COLLIN, Z., AND DOLEV, S. Self-stabilizing depth-first search. *Information Processing Letters 49*, 6 (March 1994), 297–301.

[38] COURNIER, A., DEVISMES, S., PETIT, F., AND VILLAIN, V. Snap-stabilizing depth-first search on arbitrary networks. Tech. Rep. 2004-09, 2004.

[39] COURNIER, A., DEVISMES, S., AND VILLAIN, V. Snap-stabilizing detection of cutsets. Tech. Rep. 2005-04, 2005.

[40] COURNIER, A., DEVISMES, S., AND VILLAIN, V. A snap-stabilizing dfs with a lower space requirement. Tech. Rep. 2005-05, 2005.

[41] COURNIER, A., DEVISMES, S., AND VILLAIN, V. Snap-stabilizing pif and useless computations. Tech. Rep. 2006-04, 2006.

[42] DATTA, A., GRADINARIU, M., KENITZKY, A., AND TIXEUIL, S. Self-stabilizing wormhole routing in ring networks. *Journal of Information Science and Engineering 3* (2003), 401–414.

[43] DATTA, A., JOHNEN, C., PETIT, F., AND VILLAIN, V. Self-stabilizing depth-first token circulation in arbitrary rooted network. In *5th International Colloquium on Structural Information and Communication Complexity (SIROCCO'98)* (1998), pp. 32–46.

[44] DATTA, A. K., DERBY, J. L., LAWRENCE, J. E., AND TIXEUIL, S. Self-stabilizing wormhole routing in ring networks. *Journal of Interconnexion Networks 1*, 4 (2000), 283–302.

[45] DATTA, A. K., GRADINARIU, M., AND TIXEUIL, S. Self-stabilizing mutual exclusion using unfair distributed scheduler. In Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS'00), pp. 465–470.

[46] DATTA, A. K., GRADINARIU, M., AND TIXEUIL, S. Self-stabilizing mutual exclusion under arbitrary scheduler. *The Computer Journal 47*, 3 (May 2004), 289–298.

[47] DIJKSTRA, E. W. Self-stabilizing systems in spite of distributed control. *Communications of the ACM 17*, 1 (November 1974), 643–644.

[48] DIJKSTRA, E. W. A belated proof of self-stabilization. *Distributed Computing 1*, 1 (January 1986), 5–6.

[49] DOLEV, S. optimal time self-stabilization in dynamic systems. *Lecture Notes in Computer Science 725*.

[50] DOLEV, S., GOUDA, M. G., AND SCHNEIDER, M. Memory requirements for silent stabilization. *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing* (1996), 27–34.

[51] DOLEV, S., ISRAELI, A., AND MORAN, S. Self-stabilization of dynamic systems assuming only read/write atomicity. In *Proceedings of the ninth annual ACM symposium on Principles of distributed computing* (1990), SIGOPS: ACM Special Interest Group on Operating Systems and SIGACT: ACM Special Interest Group on Algorithms and Computation Theory, ACM Press New York, NY, USA, pp. 103–117.

[52] DOLEV, S., ISRAELI, A., AND MORAN, S. Uniform dynamic self-stabilizing leader election. *IEEE Trans. on Parallel and Distributed Systems 8*, 4 (1997), 424–440.

[53] DOLEV, S., AND KAT, R. I. Self-stabilizing distributed file systems. *Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems (SRDS'02)* (2002).

[54] DOLEV, S., AND SCHILLER, E. Communication adaptive self-stabilizing communication group. Tech. Rep. TR2002-02.

[55] DOLEV, S., AND SCHILLER, E. Superstabilizing protocols for dynamic distributed systems. *Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing (Ottawa, Canada)* (1995).

[56] DOLEV, S., AND SCHILLER, E. Communication adaptive self-stabilizing group membership service. *IEEE Transactions on Parallel and Distributed Systems 14*, 7 (July 2003), 709–720.

[57] DOUGLAS, S. C. Self-stabilized gradient algorithms for blind source separation with orthogonality constraints. *IEEE-NN 11*, 6 (November 2000), 1490.

[58] DOUGLAS, S. C., KUNG, S.-Y., AND AMARI, S. A self-stabilized minor subspace rul. *IEEE Signal Proc. Lett. 5* (December 1998), 330–332.

[59] EDMOND, J., AND KARP, R. M. Theoretical improvements in algorithmic efficiency for network flow problems. *Canad J. Math 8* (1956), 399–404.

[60] FISCHER, M., LYNCH, N., AND PATERSON, M. Impossibility of distributed consensus with one faulty process. *Journal of the ACM 32*, 2 (April 1985), 374–382.

[61] FLATEBO, M., AND DATTA, A. K. Self-stabilizing load balancing for an arbitrary network. *Proceedings of the third international conference on Young computer scientists* (1993), 743–746.

[62] FLATEBO, M., AND DATTA, A. K. Two state self-stabilizing algorithms for token rings. *IEEE Trans. Software Eng. 20*, 6 (1994), 500–504.

[63] FORD, L. R., AND FULKERSON, D. W. Maximal flow through a network. 248–264.

[64] GAIRING, M., GEIST, R. M., HEDETNIEMI, S., AND KRISTIANSEN, P. Self-stabilizing algorithm for maximal 2-packing. *Nordic Journal of Computing 11*, 1 (March 2004), 1–11.

[65] GARG, V. K., AND AGARWAL, A. Self-stabilizing spanning tree algorithm with a new design methodology. Tech. Rep. TR-PDS-2004-001, 2004. available

via ftp or WWW at maple.ece.utexas.edu as technical report TR-PDS-2004-001.

[66] GÄRTNER, F. C., AND PAGNIA, H. Self-stabilizing load distribution for replicated servers on a per-access basis. In *Workshop on Self-stabilizing Systems* (1999), pp. 102–109.

[67] GHODSI, A., EL-ANSARY, S., AND KRISHNAMURTHY, S. A self-stabilizing network size estimation gossip. algorithm for peer-to-peer netwrk. Tech. Rep. T2005-16, 2005.

[68] GHOSH, AND KARAATA. A self-stabilizing algorithm for maximum matching in trees. *DISTCOMP: Distributed Computing 7* (1994).

[69] GHOSH, S. Self-stabilizing algorithms for posets on a linear array. Tech. rep.

[70] GHOSH, S., AND GUPTA, A. An exercise on fault-containment: Self-stabilizing leader election. *Information Processing Letters 59*, 5 (September 1996), 281–288.

[71] GHOSH, S., GUPTA, A., AND PEMMARAJU, M. K. S. Self-stabilizing dynamic programming algorithms on trees. In *Proceedings of the Second Workshop on Self-Stabilizing Systems* (1995), pp. 11.1–11.15.

[72] GHOSH, S., GUPTA, A., AND PEMMARAJU, S. V. A self-stabilizing algorithm for the maximum flow problem. *Distributed Computing 10*, 4 (July 1997), 167–180.

[73] GHOSH, S., AND KARAATA, M. H. A self-stabilizing algorithm for coloring planar graphs. *DISTCOMP: Distributed Computing 7* (1993), 55–59.

[74] GODDARD, G., HEDETNIEMI, S., JACOBS, D. P., AND SRIMANI, P. K. A self-stabilizing distributed algorithm for minimal total domination in an arbitrary system graph. *Proceedings of the 17th International Symposium on Parallel and Distributed Processing* (2003).

[75] GODDARD, W., HEDETNIEMI, S. T., JACOBS, D. P., AND SRIMANI, P. K. *The b-matching paper*. preprint.

[76] GODDARD, W., HEDETNIEMI, S. T., JACOBS, D. P., AND SRIMANI, P. K. Self-stabilizing distributed algorithm for strong matching in a system graph. In *Proceedings of High Performance Computing (HiPC 2003) - 10th International Conference* (Hyderabad, India, 17-20 december 2003), LNCS 2913, Springer Verlag, pp. 66–73.

[77] GODDARD, W., HEDETNIEMI, S. T., JACOBS, D. P., AND SRIMANI, P. K. Self-stabilizing global optimization algorithms for large network graphs. *International Journal of Distributed Sensor Networks 1* (2005), 329–344.

[78] GODDARD, Z. S. G., AND HEDETNIEMI, S. An anonymous self-stabilizing algorithm for 1-maximal independent set in trees. *Information Processing Letters* (2004).

[79] GOLDBERG, A., AND TARJAN, R. E. A new approach to the maximum flow problem. *J. ACM 35* (1988), 921–940.

[80] GOUDA, M., AND SCHNEIDER, M. Stabilization of maximal metric trees. *Proceedings of the Third Workshop on Self-Stabilizing Systems (published in association with ICDCS99 The 19th IEEE International Conference on Distributed Computing Systems)* (1999), 10–17.

[81] GRADINARIU, M., AND JOHNEN, C. Self-stabilizing neighborhood unique naming under unfair scheduler. *Lecture Notes in Computer Science 2150* (2001).

[82] GRADINARIU, M., AND TIXEUIL, S. Self-stabilizing vertex coloring of arbitrary graphs. *Paper presented at International Conference on Principles of Distributed Systems (OPODIS'2000) in Paris, France* (2000).

[83] HEDETNIEMI, S. T., JACOBS, D. P., AND SRIMANI, P. K. Maximal matching stabilizes in o(m) time. *Information Processing Letters 80*, 5 (2001), 221–223.

[84] HELMKE, U., AND MOORE, J. B. *Optimization and Dynamical Systems*. New York: Springer-Verlag, 1994.

[85] HERMAN, T. Probabilistic self-stabilization. *Information Processing Letters 35*, 2 (1990), 63–67.

[86] HERMAN, T., AND TIXEUIL, S. A distributed tdma slot assignment algorithm for wireless sensor networks. Tech. rep., 2004.

[87] HOWELL, R., NESTERENKO, M., AND MIZUNO, M. Finite-state self-stabilizing protocols in message-passing systems. In *Proceedings of the Third Workshop on Self-Stabilizing Systems (published in association with ICDCS99 The 19th IEEE International Conference on Distributed Computing Systems)* (1999), IEEE Computer Society, pp. 62–69.

[88] HSU, S. C., AND HUANG, S. T. Analyzing self-stabilization with finite state machine models. *Proceedings. 12th intl. conf. on dist. comp* (1992), 624–631.

[89] HSU, S.-C., AND HUANG, S.-T. A self-stabilizing algorithm for maximal matching. *Information Processing Letters 43*, 2 (1992), 77–81.

[90] HUANG, S. T. Leader election in uniform rings. *ACM Trans. Programming Languages Systems 15*, 3 (1993), 563–573.

[91] HUANG, S.-T., AND CHEN, N.-S. A self-stabilizing algorithm for constructing breadth-first trees. *Information Processing Letters 41*, 2 (February 1992), 109–117.

[92] HUANG, S. T., AND CHEN, N. S. Self-stabilizing depth-first token circulation on networks. *Distributed Computing*, 7 (1993), 61–66.

[93] HUANG, T. C., AND LIN, J.-C. A self-stabilizing algorithm for the shortest path problem in a distributed system. *Computers and Mathematics with Applications 43*, 1-2 (January 2002), 103–109.

[94] ISRAELI, S. D. A., AND MORAN, S. Resource bound for self-stabilizing message driven protocols. *SIAM Journal on Computing 26*, 1, 273–290.

[95] JOHNEN, C. Service time of self-stabilizing token circulation protocol on anonymous unidirectional rings (extended abstract).

[96] JOHNEN, C. Optimization of service time and memory space in a self-stabilizing token circulation protocol on anonymous unidirectional rings. Tech. Rep. 1330, L.R.I, September 2002.

[97] JOHNEN, C. Service time optimal self-stabilizing token circulation protocol on anonymous unidirectional rings. *21st IEEE Symposium on Reliable Distributed Systems (SRDS'02)* (2002), 80.

[98] JOHNEN, C. Bounded service time and memory space optimal self-stabilizing token circulation protocol on unidirectional rings. *Proceedings 18th International Parallel and Distributed Processing Symposium*, 26-30 (April 2004).

[99] JOHNEN, C., AND BEAUQUIER, J. Space-efficient distributed self-stabilizing depth-first token circulation. In *the Second Workshop on Self-Stabilizing Systems (Las Vegas (UNLV), USA)* (May 28-29 1995), p. 4.14.15.

[100] JOHNEN, C., AND GOUDA, M. G. Stabilizing interdomain routing in the internet. *Journal of High Speed Networks 14*, 1 (July 2005), 21–37.

[101] JOHNEN, C., AND TIXEUIL, S. Route preserving stabilization. *Lecture Notes in Computer Science 2704* (2003), 184–198.

[102] KAKUGAWA, H. Mechanical verification of self-stabilizing distributed systems. Tech. rep.

[103] KAKUGAWA, H. Uniform and self-stabilizing fair mutual exclusion on unidirectional rings under unfair distributed daemon. *Journal of Parallel and Distributed Computing 62*, 5 (May 2002), 885–898.

[104] KAKUGAWA, H., MIZUNO, M., AND NESTERENKO, M. Development of self-stabilizing distributed algorithms using transformation: case studies. In *Proceedings of the Third Workshop on Self-Stabilizing Systems* (1997), Carleton University Press, pp. 16–30.

[105] KAM, M., AND BASTAMI, F. A self-stabilizing ring protocol for load balancing in distributed real-time process control systems. Tech. Rep. UH-CS-87-8, Department of Computer Science, University of Houston, NOvember 1987.

[106] KARAATA, M. H. Self-stabilizing strong fairness under weak fairness. *IEEE Trans. on Parr. and Dist. Sys. 12*, 4 (2001), 337–345.

[107] KARAATA, M. H., AND CHAUDHURI, P. A dynamic self-stabilizing algorithm for constructing transport net. *Computing 68*, 2 (March 2002), 143–161.

[108] KATZ, S., AND PERRY, K. J. Self-stabilizing extensions for message-passing systems. In *Proceedings of the ninth annual ACM symposium on Principles of distributed computing* (1990), SIGOPS: ACM Special Interest Group on Operating Systems and SIGACT: ACM Special Interest Group on Algorithms and Computation Theory, ACM Press New York, NY, USA, pp. 91–101.

[109] KO, B.-J. Distributed, self-stabilizing placement of replicated resources in emerging networks. *11th IEEE International Conference on Network Protocols (ICNP'03)* (2003).

[110] KRUIJER, H. S. M. Self-stabilization (in spite of distributed control) in tree-structured systems. *Information Processing Letters 8* (January 1979), 91–95.

[111] KULKARNI, S. S., AND ARUMUGAM, U. Collision-free communication in sensor networks. *Proceedings. 6th International symposium of self-stabilizing systems* (2003), 17–31.

[112] LIN, J.-C., AND HUANG, T. C. An efficient fault-containing self-stabilizing algorithm for finding a maximal independent set. In *IEEE Transactions on Parallel and Distributed Systems* (August 2003), vol. 14, pp. 742–754.

[113] MIZUNO, M., AND NESTERENKO, M. A transformation of self-stabilizing serial model programs for asynchronous parallel computing environments. *Information Processing Letters 66*, 6 (1998), 285–290.

[114] MIZUNO, M., NESTERENKO, M., AND KAKUGAWA, H. Lock based self-stabilizing distributed mutual exclusion algorithms. In *International Conference on Distributed Computing Systems* (1996), pp. 708–716.

[115] NESTERENKO, M., AND MIZUNO, M. A quorum-based self-stabilizing distributed mutual exclusion algorithm. *Journal of Parallel and Distributed Computing 62*, 2 (February 2002), 284–305.

[116] OEHLERKING, J., DHAMA, A., AND THEEL, O. Towards automatic convergence verification of self-stabilizing algorithms. *7th International Symposium on Self-Stabilizing Systems(SSS2005), Barcelona, Spain* (October 2005).

[117] OJA, E., AND KARHUNEN, J. On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix. *J. Math Anal. Appl 106*, 1 (1985), 69–84.

[118] PAPATRIANTAFILOU, M., AND TSIGAS, P. *Self-Stabilizing Wait-Free Clock Synchronization.* Centrum voor Wiskunde en Informatica (CWI), ISSN 0169-118X, 31 1994.

[119] PETIT, F. Highly space-efficient self-stabilizing depth-first token circulation for trees. In *OPODIS'97, International Conference On Principles Of Distributed Systems Proceedings* (1997), pp. 221–235.

[120] PETIT, F. *Efficiency and Simplicity in Self-Stabilizing Distributed Depth-First Token Circulation Algorithms*. PhD thesis, Universite de Picardie Jules Verne, TR98-05, LaRIA, Amiens France, 1998.

[121] PETIT, F., AND VILLAIN, V. Color optimal self-stabilizing depth-first token circulation protocol for asynchronous message-passing. In *PDCS-97 10th International Conference on Parallel and Distributed Computing Systems Proceedings* (1997), International Society for Computers and Their Applications, pp. 227–233.

[122] PETIT, F., AND VILLAIN, V. Color optimal self-stabilizing depth-first token circulation protocol for asynchronous message-passing. In *PDCS-97 10th International Conference on Parallel and Distributed Computing Systems Proceedings* (1997), International Society for Computers and Their Applications, pp. 227–233.

[123] PETIT, F., AND VILLAIN, V. Time and space optimality of distributed depth-first token circulation algorithms. In *DIMACS Workshop on Distributed Data and Structures (Princeton, USA)* (May 10-11 1999), pp. 91–106.

[124] QADEER, S., AND SHANKAR, N. Verifying a self-stabilizing mutual exclusion algorithm. In *IFIP International Conference on Programming Concepts and Methods (PROCOMET '98)* (Shelter Island, NY, 1998), D. Gries and W.-P. de Roever, Eds., Chapman & Hall, pp. 424–443.

[125] SAIFULLAH, A. 2-edge-connectivity and 2-vertex-connectivity with fault containment. Tech. Rep. WUCSE-2011-96. Washington University in St Louis.

[126] SAIFULLAH, A., AND TSIN, Y. A self-stabilizing algorithm for 3-edge-connectivity. In *The 5th International Symposium on Parallel and Distributed Processing and Applications (ISPA 2007)*. Lecture Notes in Computer Science. vol. 4742. p. 6-19).

[127] SAIFULLAH, A., AND TSIN, Y. A self-stabilizing algorithm for 3-edge-connectivity, 2011.

[128] SAIFULLAH, A., AND TSIN, Y. Self-stabilizing computation of 3-edge-connected components, 2011.

[129] SCHNEIDER, M. Self-stabilization. *ACM Computing Surveys 25*, 1 (March 1993), 45–67.

[130] SELIC, B. Fault tolerance techniques for distributed systems. *(Staff, IBM)* (July 2004).

[131] SHEN, Z., AND TIRTHAPURA, S. A snap-stabilizing dfs with a lower space requirement. Tech. Rep. TR-2004-04-4, April 2004.

[132] SHUKLA, S., ROSENKRANTZ, D., AND RAVI, S. Developing self-stabilizing coloring algorithms via systematic randomization. *Proceedings of the International Workshop on Parallel Processing* (1994), 668–673.

[133] SHUKLA, S., ROSENKRANTZ, D., AND RAVI, S. Observations on self-stabilizing graph algorithms for anonymous networks. In *Proceedings of the Second Workshop on Self-Stabilizing Systems* (1995), pp. 7.1–7.15.

[134] SUR, S., AND SRIMANI, P. K. A self-stabilizing distributed algorithm to construct bfs spanning trees of a symmetric graph. *Parallel Processing Letters 2*, 2-3 (1992), 171–179.

[135] SUR, S., AND SRIMANI, P. K. A self-stabilizing algorithm for coloring bipartite graphs. *Information Processing Letters 69* (1993), 219–227.

[136] TEL, G. Introduction to distributed algorithms (2nd edition). *Cambridge University Press* (2001).

[137] XU, Z., HEDETNIEMI, S. T., GODDARD, W., AND SRIMANI, P. K. A synchronous self-stabilizing minimal domination protocol in an arbitrary network graph. In *Proceedings of the 5th International Workshop on distributed computing(IWDC)* (27-30 December 2003), LNCS 2918, pp. 26–32.

[138] YEN, I.-L. A highly safe mutual exclusion self-stabilizing algorithm. *Information Processing Letters 57*, 6 (March 1996), 301–305.