Brief Announcement: Computing automatically the stabilization time against the worst and the best schedules

Joffroy Beauquier, Colette Johnen, Stéphane Messika

L.R.I./C.N.R.S., Université Paris-Sud 11

Abstract: In this paper, we reduce the problem of computing the convergence time for a randomized self-stabilizing algorithm to an instance of the stochastic shortest path problem (SSP). The solution gives us a way to compute automatically the stabilization time against the worst and the best policy. Moreover, a corollary of this reduction ensures that the best and the worst policy for this kind of algorithms are memoryless and deterministic. We apply these results here in a toy example. We just present here the main results, to more details, see [1].

By their very nature, distributed algorithms have to deal with a non-deterministic environment. Speeds of the different processors or the message delays are generally not known in advance and may vary substantially from one execution to the other. For representing the environment in an abstract way, the notion of scheduler (also called demon or adversary) has been introduced. The scheduler is in particular responsible of which processors take a step in a given configuration or of which among the messages in transit arrives first. It is well known that the correctness of a distributed algorithm depends on the considered scheduler. This remark also holds for self-stabilizing distributed algorithm.

In this paper, we restrict our attention to probabilistic self-stabilization. Classically self-stabilization requires convergence (each execution reaches a legitimate configuration) and correctness (each execution starting from a legitimate configuration satisfies the specification). Probabilistic self-stabilization requires that convergence is probabilistic. It appears that the convergence property of a given algorithm depends on the chosen policy. With some policies the algorithm can converge in a finite number of steps (the stabilization time) while with others it can not converge at all. Even if the stabilization time is finite, it can differ according to the policy. It is thus interesting to know the best policy (the policy that gives the smaller expected stabilization time) and the worst. Note that the best policy can possibly give a finite stabilization time and the worst an infinite one. In some cases best and worst both give finite stabilization time (it is then said that the algorithm is self-stabilizing under the distributed scheduler: scheduler that produce any k-bounded policy).

In a distributed system, all the machines are finite state machines. A configuration X of the distributed system is the N-tuple of all the states of the machines. The code is a finite set of guarded rules. The guard of a rule on p is a boolean expression involving the state of p and its neighbors. A machine p is *enabled* in a configuration c, if a rule guard of p is true, in c. The execution simultaneously by several enabled machines of rules is call a *computation step*.

A randomized distributed algorithm can be seen as a Markov Decision Process. Informally, a Markov Decision Process is a generalization of a Markov chain in which a set of possible actions is associated to each state. To each state-action pair corresponds a probability distribution on the states, which is used to select the successor state. A Markov chain corresponds thus to a Markov Decision Process in which there is exactly one action associated with each state. The formal definition is as follows.

Definition 1 (Markov Decision Process). *A Markov Decision Process (MDP)* $(S,\ Act,\ A,\ p)$ *consists of a finite set* S *of states, a finite set* Act *of actions, and two components* $A,\ p$ *that specify the transition structure.*

- *For each* $s \in S$, $A(s)$ *is the non-empty finite set of actions available at* s.
- *For each* $s, t \in S$ *and* $a \in A(s)$, $p_{st}(a)$ *is the probability of a transition from* s *to* t *when action* a *is selected. Moreover,* p *verifies the following property* $\forall s, \forall a \in A(s)$ *we have* $\sum_{t \in S} p_{st}(a) = 1$.

The MDP associated with a distributed algorithm is defined by (i) S is the set of configurations, (ii) Act is the set of machine sets, (iii) $A(c)$ is the set of enabled machines in c, (iv) $p_{st}(a)$ is the probability to reach the configuration t from a configuration s by a computation step where all processors in a execute a rule.

Policies are closely related to the adversaries of Segala and Lynch, and to the schedulers of Lehman and Rabin, Vardi, Pnueli and Zuck , and to the strategies of Beauquier, Delaët, Granidariu, and Johnen.

Definition 2 (Policy). *A policy η is a set of conditional probabilities $Q_\eta(a|s_0 s_1 ... s_n)$, for all $n \geq 0$, all possible sequences of states $s_0, ..., s_n$ and all $a \in A(s_n)$ such that $0 \leq Q_\eta(a|s_0, s_1 ..., s_n) \leq 1$ and $\sum_{a \in A(s_n)} Q_\eta(a|s_0, s_1 ..., s_n) = 1$.*

Definition 3 (Probability measure under a policy). *Let η be a policy. Let $h = s_0 a_0 s_1 a_1 ... s_n$ be a sequence of computation steps. $P_s^\eta(C_h) = \prod_{k=0}^{n-1} p_{s_k s_{k+1}}(a_k) Q_\eta(a_k|s_0, s_1 ..., s_k)$.*

Under a given policy, the randomized distributed algorithm can be seen as a Markov chain. In this Markov chain, we denoted by X_n the configuration reached after n computation steps (X_n is the random variable).

Definition 4 (Probabilistic convergence). *Let Leg be the legitimate predicate defined on configurations. A probabilistic distributed algorithm A under a policy η probabilistically converges to Leg iff : from any configuration s, the probability to reach a legitimate configuration is equal to 1 under the policy η. Formally, $\lim_{n \to \infty} P_c^\eta(\exists m \leq n \mid X_m \in L) = 1$ where X_m is the reached state after m computation steps in the Markov chain defined by : MDP associated to A, c and η.*

The convergence time under a policy is the expectation value of the random variable Y under this policy (Y being the number of computation steps to reach a legitimate configuration). The best policy (resp. the worst policy) is the policy having the smallest (resp. largest) convergence time.

Informally, the Stochastic Shortest Path problem consists in computing the minimum expected cost for reaching a given subset of destination states, from any state of a Markov Decision Process in which a cost is associated to each action.

Definition 5 (Instance of Stochastic Shortest Path Problem). *An instance of the stochastic shortest path problem is (M, U, c, g) in which (i) M is a Markov Decision Process, (ii) U is the set of destination states, (iii) c is the cost function, which associates to each state $s \in S - U$ and action $a \in A(s)$ the cost $c(s, a)$, and (iv) g is the terminal cost function which associates to each $s \in U$ its terminal cost $g(s)$.*

In [1], we have define two instances of SSP (called SSP_b and SSP_w). SSP_b allows us to compute the best convergence time. SSP_w is designed to compute the worst convergence time.

- SSP_b: M is the MDP associated to the algorithm. U is the set of legitimate configurations. c is always equal to 1. g is function null.
- SSP_w: M is the MDP associated to the algorithm. U is the set of legitimate configurations. c is always equal to -1. g is function null.

Definition 6 (Bellman operator). *Let (M, U, c, g) be an instance of the SSP problem. We denote $v = (v_s)_{s \in S-U}$ a vector of real numbers. We define the Bellman operator L by*

$$L(v_s) = min_{a \in A(s)} \{c(s, a) + \sum_{t \in S-U} p_{st}(a) v_t + \sum_{t \in U} p_{st}(a) g(t)\}$$

Theorem 1. *Computing the convergence time for a randomized self-stabilizing algorithm under the best and worst policy can be reduced to an instance of SSP.*

Thus, one can apply the result on the SSP problem. For instance, the Bellman operator has a fixpoint in SSP_b and in SSP_w. Thus, the convergence time under the best and worst policy can be compute automatically (via the fixpoint of Bellman operator). We can also obtain the corresponding policies. In SSP_b, (resp. SSP_w) on each configuration, the selected action is the best choice (resp. worst choice) to converge from this configuration.

Algorithm 1 token circulation on anonymous and unidirectional rings

Constant:
N is the ring size. m_N is the smallest integer not dividing N.
Variables on p: v_p is a variable taking value in [0, m_N -1].
Random Variables on p:
$rand\_bool_p$ taking value in {1, 0}. Each value has a probability 1/2.
Action on p: lp is the clockwise preceeding neighbor of p
$\mathcal{R}$:: $(v_p - v_{lp} \neq 1 \bmod m_N) \rightarrow$ if $rand\_bool_p = 1$ then $v_p := (v_{lp} + 1) \bmod m_N$;
.

Toy Example The presented algorithm achieves token circulation on unidirectional ring, it was defined by Beauquier and al. in [BDC95].
A processor is said to have a token iff it is enabled. The algorithm to be self-stabilizing should converge from a configuration with several tokens to a configuration with one token. Notice that the algorithm ensures that in any configuration, the ring has at least one token.
Configurations are gathered in class. Configurations in which the tokens are at the same (clockwise) distance d, belong to the same class denoted d. If a token $T1$ is at distance d of the other token $T2$ then $T2$ is at distance N-d of $T1$. Therefore class d and N-d are identical. Notice that if $d = 0$ then the ring has only one token. The best convergence time is easy to guess here, from the configuration of the class $0 < d \leq N/2$ is $2 \times d$, whatever is the ring size. This convergence time is achieved under the following policy: in any configuration, the token at distance d of the other token tries to catch up the unmoving token.
Let us take $N = 5$, then there are 3 classes c_0, c_1, c_2 There are also three kinds of policy in each different configuration, one can choose only the closest token (action c), the furthest (f), or both of them (b). In the corresponding SSP instance we have $U = c_0$ and we can start from the vector $v^0 = (0, 0)$ then applying one time the Bellman operator L we obtain that $v^1 = (1, 1)$ then $v^2 = (3/2, 2)$, $v^3 = (7/4, 11/4)$, $v^4 = (15/8, 26/8)$, $v^5 = (31/16, 57/16)$, $v^6 = (63/32, 120/32)$, $v^7 = (127/64, 247/64)$, $v^8 = (255/128, 502/128)$ and the sequence converges towards $v^\bullet = (2, 4)$ which is effectively a fixpoint. Then, by getting the action in which the minimum is reached we obtained that the best policy is the one that chose always action c, that is conform to the intuition.

In [1], we study two other self-stabilizing algorithms : vertex coloring, and naming in grids. There is a self-stabilizing deterministic algorithm giving distinct name to nodes in grids. Using this technique we show that the convergence time under the worst and best policy are better with our randomized algorithm that with the deterministic one.

One could think that the best policy and the worst policy are intricate and difficult to describe, or that their simulation would use a lot of resources. In fact it is not true because there are always a best policy and a worst policy that are memoryless (meaning that the choice they make in a given configuration depends only on the configuration, and not on the past of the execution). This results extend a result in [3] and, although in a different context, a result of [2].

Corollary 1. *The best and the worst policy (considering the convergence time) for a randomized self-stabilizing algorithm are memoryless and deterministic.*

This last result allows us to only consider memoryless policies when studying self-stabilizing algorithms. Indeed, the worst convergence time is always given by a memoryless policy. This result considerably simplifies the verification of the correctness and the computation of the complexity of these algorithms.

References

1. J. Beauquier, C. Johnen, and S. Messika. Computing automatically the stabilization time against the worst and the best schedulers. Technical Report 1448, L.R.I, 2006.
2. D.P. Bertsekas and J.N. Tsitsiklis. An analysis of stochastic shortest path problems. *Math of Op. Res.*, 16(2):580–595, 1991.
3. L. de Alfaro. *Formal Verification of Probabilistic systems.* PhD Thesis, Stanford University, 1997.