Decision-Making with Cross-Entropy for Self-Adaptation

Gabriel A. Moreno*, Ofer Strichman*†, Sagar Chaki* and Radislav Vaisman‡
*Software Engineering Institute, Carnegie Mellon University, Pittsburgh, USA
†Information Systems Engineering, IE, Technion, Haifa, Israel
‡School of Mathematics and Physics, University of Queensland, Australia
gmoreno@sei.cmu.edu, ofers@ie.technion.ac.il, chaki@sei.cmu.edu, slvaisman@gmail.com

***Abstract*—Approaches to decision-making in self-adaptive systems are increasingly becoming more effective at managing the target system by taking into account more elements of the decision problem that were previously ignored. These approaches have to solve complex optimization problems at run time, and even though they have been shown to be suitable for different kinds of systems, their time complexity can make them excessively slow for systems that have a large adaptation-relevant state space, or that require a tight control loop driven by fast decisions. In this paper we present an approach to speed up complex proactive latency-aware self-adaptation decisions, using the *cross-entropy* (CE) method for combinatorial optimization. The CE method is an any-time algorithm based on random sampling from the solution space, and is not guaranteed to find an optimal solution. Nevertheless, our experiments using two very different systems show that in practice it finds solutions that are close to optimum even when its running time is restricted to a fraction of a second, attaining speedups of up to 40 times over the previous fastest solution approach.**

***Keywords*-cross-entropy method; decision-making; optimization, self-adaptive systems**

I. Introduction

Self-adaptive systems are able to change their structure and/or behavior to deal with changes in their environment so that they continue to satisfy their requirements or perform as best as they can [1], [2]. An important aspect of these systems is deciding if they have to adapt and how to do it. Approaches to decision-making in self-adaptive systems are increasingly becoming more effective at managing the target system by taking into account more elements of the decision problem that were previously ignored, such as environment uncertainty [3], [4], multiple sources of uncertainty [5], probabilistic outcomes of adaptation actions [6], [7], [8], and adaptation timing [9]. These approaches have to solve complex optimization problems at run time, and many of them rely on the use of probabilistic model checking to do so. Even though they have been shown to be suitable for different kinds of systems, their time complexity can make them excessively slow for systems that have a large adaptation-relevant state space, or that require a tight control loop driven by fast decisions. For that reason, there has been work to make the use of some of these techniques more practical, for example, by using caching, precomputation, and near-optimality [10], or by performing as much of the computation as possible off-line [11].

In this paper we present another approach to speed up complex self-adaptation decisions, using the *cross-entropy* (CE) method for combinatorial optimization [12]. The CE method attempts to solve an optimization problem by randomly sampling from the solution space, and iteratively updating the sampling distribution so that the probability of finding an optimal solution increases in each iteration. In particular, we use the CE method to speed up proactive latency-aware (PLA) adaptation decisions. A PLA decision solves an optimization problem of finding the sequence of adaptation actions that should be executed over a finite horizon in order to maximize the expected utility that the system could obtain, given a probabilistic model of the near-future environment behavior. Specifically, we make three contributions.

First, we show how the PLA optimization problem can be solved via the CE method. The main technical challenge here is that the solution set of the PLA optimization problem is sparse, i.e., not all sequences of adaptation actions are legal candidate solutions. This is because the set of legal candidate adaptation actions valid at a given time depends on the state of the target system, which is also affected by previous adaptations. However, the classical CE method was designed to solve problems with *dense* solution spaces, such as MAX-SAT, where the goal is to find a variable assignment that satisfies the maximum number of clauses of a propositional formula expressed in conjunctive normal form. In this case, all variable assignments are legal candidate solutions. We address this challenge by developing a variant of the CE method that efficiently samples the sparse solution space of PLA adaptation decisions.

Second, we implement our approach, PLA-CE, in an adaptation manager and validate it on two examples of self-adaptive systems from two distinct domains: (i) DART consists of a group of communicating and collaborating quadcopters that are trying to achieve a mission under uncertain operational conditions; this example comes from the distributed cyber-physical domain; and (ii) SWIM simulates a web-application with a dynamically adjustable number of servers that process requests aiming to achieve acceptable quality of service requirements; this example comes from a more traditional IT domain. In this way, we expect our validation to span a broader class of systems.

Finally, we empirically show the advantage of PLA-CE over

competing techniques in solving the PLA adaptation decision problem. Specifically, using the CE method, we obtain a substantive speedup—of up to 40 times—with respect to PLA-SDP [11], which is currently the best PLA solution. Indeed, PLA-SDP has been already been shown [11] to be an order of magnitude faster than PLA-PMC, the PLA approach based on probabilistic model checking [9]. In addition, we evaluate the effectiveness of adaptation decisions with PLA-CE, since the CE method is an any-time algorithm based on sampling, and is not guaranteed to find an optimal solution. In spite of this, our experiments show that in practice it finds solutions that are close to optimum even when restricted to short timeouts.

Even though the CE method for optimization is more than a decade old, it has not been previously used for decision-making in self-adaptive systems. In this paper we focus on the application of the CE method to PLA. However, we conjecture it would be possible to use it to speed up other complex decision approaches for self-adaptive systems where finding the optimal solution is not an option because of the time constraints required for run-time decision-making.

The rest of the paper is organized as follows. In Section II we provide some background on proactive latency-aware adaptation and the cross-entropy method for optimization. The PLA-CE adaptation decision approach is presented in Section III. In Section IV, we present the evaluation of the approach. Related work is in Section V, and our conclusions are in Section VI.

II. Background

In this section, we introduce the two main areas relevant to our work—proactive latency-aware adaptation and the cross-entropy method for optimization.

A. Proactive Latency-Aware Adaptation

Typically, self-adaptation approaches rely on a set of adaptation tactics that they can use to deal with different conditions. For example, adding a server is a tactic that can be used to deal with increased load in the system, and revoking permissions from a user is a suitable tactic for protecting the system from an insider attack. There are tactics that can take some non-trivial time to execute, and this is a fact that is not considered by most self-adaptation approaches. For example, provisioning a new virtual machine in the cloud can take a few minutes [13]. We refer to the period of time between when a tactic is started and when its effect is produced as *tactic latency*.

Proactive latency-aware adaptation improves self-adaptation effectiveness over reactive approaches by considering both the current and anticipated adaptation needs of the system, and taking into account the latency of adaptation tactics [9], [11]. With latency awareness, it explicitly considers how long adaptation tactics take to execute, both to account for the delay in producing their effect, and to avoid solutions that are infeasible when the time dimension is considered (e.g, one that assumes having one more server immediately). To be proactive, PLA leverages knowledge or predictions about the future states of the environment to start adaptation tactics with the necessary lead time so that they can complete on time, and to avoid unnecessary adaptations. In addition, it supports concurrent tactic execution, exploiting nonconflicting tactics to speed up adaptations that involve multiple tactics, and to complement long-latency tactics with faster ones that can produce intermediate results sooner.

We assume that the adaptation goal is *maximizing aggregate utility* over the execution of the system. Utility, for example, can be defined by a service level agreement (SLA) that specifies rewards and penalties for meeting response time and quality requirements in a system, or it can be the number of targets detected in a surveillance mission. Given that adaptation goal, the adaptation decision answers the question of what adaptation tactic(s) should be started now, if any, to maximize the aggregate utility that the system will provide in the rest of its execution. However, the adaptation actions the system takes now can constrain its available actions in the near future. For example, starting a tactic with latency may prevent other conflicting tactics from being used in the meantime. This means that the adaptation decision has to consider not only the current state of the system and the environment, but also how both will evolve. This requires predictions of the near future state of the environment, and since the further they are into the future, the more uncertainty they have, the decision is limited to a finite lookahead horizon. Making adaptation decisions is then a problem of selecting adaptation actions in the context of the probabilistic behavior of the environment, such that the utility accumulated over the decision horizon is maximized. Therefore, PLA uses a Markov Decision Process (MDP), which is a model for sequential decision making when the outcome of taking an action in a given state is uncertain [14], with the uncertainty in this case arising from the uncertain environment.

PLA makes these adaptation decisions periodically, with a fixed decision interval. Before each decision, the underlying MDP is updated with the latest environment predictions over the decision horizon, which is discretized with the same interval. A policy for an MDP dictates what actions to take in each state. The optimal policy, in particular, maximizes the expected aggregate utility over the decision horizon. After the optimal policy is computed, PLA commits only to the first action, which it starts executing, and ignores the rest of the policy, since it will recompute a new policy in the following decision period.

B. The Cross-Entropy Method for Optimization

The CE method is a powerful technique for solving difficult estimation and optimization problems, based on Kullback-Leibler (or cross-entropy) minimization [15]. It was introduced by Rubinstein in 1999 [16] as an adaptive sampling procedure for the estimation of rare-event probabilities (the point being that the probability of rare events cannot be efficiently estimated by simple Crude Monte Carlo sampling [17]). Subsequent work in [18], [19] has shown that many optimization queries can be translated into rare-event estimation problems. The reason is that the optimal solution is a rare-event in itself,

and hence finding it with naive random search is not likely to succeed.

The CE method gradually changes the sampling distribution of the random search, so that the rare-event is more likely to occur. It develops a sequence of parametric sampling distributions that converges (asymptotically [20]), to a desired distribution with probability mass concentrated in a region of near-optimal solution. In order to develop such a sequence, it uses the Kullback-Leibler divergence [21] as a measure of closeness between two distributions.

To date, the CE method has been successfully applied to many different optimization and estimation problems. The former includes mixed integer non-linear programming [22]; continuous optimal control problems [23], [24]; continuous multi-extremal optimization [25]; multidimensional independent component analysis [26]; optimal policy search [27]; clustering [28], [29], [30]; signal detection [31]; DNA sequence alignment [32], [33]; fiber design [34]; noisy optimization problems such as optimal buffer allocation [35]; resource allocation in stochastic systems [36]; network reliability optimization [37]; vehicle routing optimization with stochastic demands [38]; power system combinatorial optimization problems [39]; and neural and reinforcement learning [40], [41], [42], [43]. CE has even been used as a main engine for playing games such as Tetris, Go and backgammon [44]. See [45] and [15] for further details.

To understand how CE is used in the context of our problem, we describe it as solving a general discrete constrained optimization problem, which can be formally stated as follows. We are given a set of variables $v_1, \ldots, v_n$ with domains $D_1, \ldots, D_n$, respectively, a set of constraints over the variables, and an objective function. The goal is to find an assignment to the variables from their respective domains that respects the constraints and maximizes the objective. When there is not enough time to solve this problem, CE gives us an effective approximation, since it is an *any-time* algorithm that typically finds good solutions in a short amount of time.

CE is an adaptive sampling method. We denote by F_i the Probability Distribution Function (PDF) of v_i, which is initially set to be uniform over D_i. That is, initially the value of v_i is chosen at random with equal probability to each of D_i's elements. In each sampling round (lines 2–8 in Alg. 1) N samples are generated, and in each of those, all variables are assigned a value from their respective domain, according to the F_i distributions. The overall solution is then evaluated assigning it a score such that the better the solution is, the higher its score is (line 6). In addition, if a sample is better than all those seen so far, then its corresponding assignment is saved in a variable *BestSolution*, in line 7.

After completing N samples, in line 9, the best $\lceil \rho * N \rceil$ samples are selected, for some $\rho \in [0, 1]$. In other words, ρ is the ratio of the best results that is used for adapting the sampling distributions. The selected set of best samples is called the "elite set". Based on this set, the distributions F_i are adjusted. For example, if D_1 has ten values $d_1, \ldots, d_{10}$, and in the elite set, d_2, d_5, and d_8 were selected 40%, 20%, and 40% of the time, respectively, then a new distribution F'_1 is defined with these numbers, i.e.,

$$F'_1(d_2) = 0.4, F'_1(d_5) = 0.2, F'_1(d_8) = 0.4$$

and for $1 \le i \le 10, i \notin \{2, 5, 8\}$ we have $F'_1(d_i) = 0$. Next, this new distribution is "blended" with previous one via *exponential smoothing*, in line 10. The parameter $\alpha \in [0, 1]$ denotes the exponential smoothing factor. The two extreme values of α, namely 0 and 1, imply ignoring the new distribution F'_i and ignoring the history, respectively. Hence in practice it makes sense to choose some number in between, where the larger α is, the more weight we give to the new sample.

The algorithm stops and returns the best solution seen, when either it *converges* or it exhausts the time, or number of iterations, that we allocate to it.[1] Convergence means that for each variable v_i,

$$\exists d \in D_i \,.\, 1 - \epsilon \le F_i(d) \le 1$$

where ϵ is a parameter of the algorithm called the *convergence threshold*. Typically it is set to a positive number close to 0. In other words, convergence means that only with a very small probability the next sample will be different than the previous ones.

Algorithm 1 The cross-entropy method.

1: **while** (!converged and !timeout) **do**
2:     **for** $j = 1..N$ **do**
3:         **for** $i = 1..n$ **do**
4:             choose value for v_i randomly according to F_i;
5:         **end for**
6:         Evaluate solution;
7:         Update *BestSolution* if applicable;
8:     **end for**
9:     Choose $\lceil \rho * N \rceil$ best solutions to form distribution F'_i;
10:    $F_i = \alpha F'_i + (1 - \alpha) F_i$;
11: **end while**
12: **return** *BestSolution*;

Finally, let us go back to line 4, where a value is selected according to a given PDF F_i. The technique for making this selection can be described in several steps. First, we calculate the Cumulative Distribution Function (CDF) for F_i. That is, if $D_i = \{d_1, \ldots, d_n\}$ then for $1 \le j \le n$

$$CDF_i(j) = \Sigma_{l=1..j} F_i(d_l) \,. \tag{1}$$

As a special case, $CDF_i(0) = 0$. Then, we sample uniformly a number u in the range $[0..1]$, and search for the smallest j such that:

$$(1 \le j \le n) \wedge (CDF_i(j-1) \le u < CDF_i(j))$$

This is done with binary search, and is hence highly efficient.

[1] In Alg. 1, a timeout is used as a stopping condition. However, in our setting we do not specify the timeout explicitly, rather we limit the number of iterations.

III. Adaptation Decisions with Cross-Entropy

In this section, we present PLA-CE, an approach to making self-adaptation decisions using the cross-entropy method. The adaptation decision is done periodically, with an interval between decisions of duration τ. The goal of the decision is to find the adaptation tactic(s) that should be started in order to maximize the expected utility over the decision horizon. Doing that with the CE method requires being able to generate random solutions according to the sampling distributions, and being able to evaluate them.

A. Solution Generation

The decision horizon has H periods of duration τ, and at the beginning of each period, the system can execute an *adaptation action*, which is a (possibly empty) set of adaptation tactics that can be started concurrently. Even though the result of the adaptation decision is only the adaptation action to be taken at the beginning of the first period, evaluating a solution requires computing the utility that it would achieve over the whole decision horizon. Since the system can keep adapting in subsequent decision periods, the evaluation must also take into account the adaptation actions that the system would take after the first one. Therefore, a solution is a sequence of adaptation actions $\langle a_1, \ldots, a_H \rangle$, referred to as an *adaptation path*.

A random solution can be generated by drawing action a_i according to F_i, for $i = 1, \ldots, H$. However, it is possible to end up with an infeasible solution, given the applicability conditions of the adaptation actions. The system configuration[2] at the time the decision is being made is c_0. Therefore, an adaptation path is feasible only if a_1 is applicable in system state c_0. Furthermore, even if a_1 is valid, when applied it will take the system from configuration c_0 to c_1, which in turns limits the feasible actions for a_2 to those applicable in c_1, and so on. One way to deal with infeasible solutions is to assign them an extremely low value. However, in Section III-C, we introduce an optimization that generates only feasible solutions.

B. Solution Evaluation

The evaluation of a solution has to consider the joint evolution of the system and the environment over the decision horizon to determine how much utility it would attain. The utility function $U(c, e)$ denotes the utility that the system would attain during the period between decisions if it has configuration c and the environment is in state e.

Given a solution, the sequence of configurations that the system will go through over the decision horizon can be computed by applying the adaptation actions in the path, starting with the current configuration c_0. To do that, we need a model of the transitions of the system. PLA-SDP, one of the PLA solution approaches, computes off-line the feasible transitions in the system MDP using formal models and analysis [11]. The result of this off-line computation comprises two kinds of reachability predicates that model the adaptation process:

- $R^I(c, c')$ indicates that configuration c' can be reached *immediately* from c through the use of an adaptation action. The action that realizes this configuration change can be obtained using a partial mapping $Act^I : C \times C \nrightarrow A$, where C is the set of all possible configurations and A is the set of all adaptation actions. The mapping Act^I is also computed offline. Even for tactics with latency, the start of the tactic is an immediate configuration change, from one in which the tactic is not running to one in which it is, since tactic progress is part of the system state.
- In addition, there is *delayed reachability*, where $R^D(c, c')$ indicates that if the system is in configuration c, with the passage of one decision interval it will reach configuration c'. That is, R^D models the transitions that happen due to the execution of tactics with latency. Note that in this case, c and c' do not necessarily represent configurations at the beginning and end of the execution of an adaptation tactic, since the transition delay modeled by R^D is a single period of duration τ, whereas the latency of the tactic could be longer. When that is the case, there are intermediate delayed transitions that represent the progress of the execution of the tactic.

Since configuration changes are deterministic under the execution of an adaptation action, or the passage of time, using the predicates R^I and R^D and the mapping Act^I that PLA-SDP computes, we can define the following functions that give us the configuration that is reached starting from a given configuration and either applying an adaptation action or letting one period of time pass, respectively.

$$P^I : C \times A \nrightarrow C \tag{2}$$

$$P^D : C \rightarrow C \tag{3}$$

where P^I is a partial function because some actions are not applicable in some system states. More specifically, we have:

$$P^I(c, a) = c' \iff R^I(c, c') \wedge Act^I(c, c') = a$$
$$P^D(c) = c' \iff R^D(c, c')$$

The system adaptation over the decision horizon can be modeled as an initial immediate action taken at the time the decision is made followed by a sequence of alternating delayed and immediate transitions. The delayed transition happens over one period, and is followed by an immediate transition that happens as soon as the delayed one completes. This models the adaptation over the decision horizon, in which an adaptation decision is made at the beginning of each time period. Given this model, the sequence of configurations $c_1, \ldots, c_H$ that results from applying a candidate solution $\langle a_1, \ldots, a_H \rangle$ to the current configuration c_0 can be computed as

$$c_1 = P^I(c_0, a_1) \tag{4}$$

$$c_t = P^I\left(P^D(c_{t-1}), a_t\right), \quad t > 1 \tag{5}$$

[2]We use *system configuration* and *system state* interchangeably.

If the future evolution of the environment was known, it would be possible to compute the value of a solution as[3]

$$v = \sum_{t=1}^{H} \hat{U}(c_t, e_t) \ .$$

where e_t denotes the environment state at time interval t. However, in general, we only have predictions of future environment states over the decision horizon, and these are subject to uncertainty. The uncertain environment can be modeled as a stochastic process in which the random variable representing the state of the environment has one realization at each time step, with a time step being equal to the decision period τ. In particular, PLA uses discrete-time Markov chains (DTMCs) to model the probabilistic behavior of the environment, with the probability of transitioning from state e to state e' in one time period given by $p(e' \mid e)$. Additionally, E_t denotes the set of environment states feasible in time interval t.

As long as the environment model is encoded in this way, there is no particular requirement for how the model is constructed, or what particular topology the DTMC has to encode. For example, for one of the systems used for evaluation, we use a time series predictor, and based on past environment states, we construct, at the time the decision is made, a probability tree for the behavior of the environment over the decision horizon following the procedure described by Moreno et al. [9].

Having the sequence of system configurations that would result from the candidate solution, and the environment model, we can compute the value v of the solution required for line 6 of the algorithm with backward induction as follows

$$v^H(c_H, e) = \hat{U}(c_H, e), \quad \forall e \in E_H \tag{6}$$

$$v^t(c_t, e) = \hat{U}(c_t, e) + \sum_{e' \in E_{t+1}} p(e' \mid e) v^{t+1}(c_{t+1}, e'), \quad \forall e \in E_t,\ t = H-1, \ldots, 1 \tag{7}$$

$$v = \sum_{e' \in E_1} p(e' \mid e_0) v^1(c_1, e') \tag{8}$$

where e_0 is the current state of the environment. First, the value of having configuration c_H for each of the possible realizations of the environment at the end of the horizon is computed with (6). The rest of the intermediate valuations v^t are computed with (7), considering both the utility obtained in the period t, and expected utility that can be obtained afterwards given how the environment can evolve in subsequent periods if the environment state is e. The valuation v of the complete solution is given by (8), which computes the expected value over the probability distribution of the environment evolution given the current state of the environment e_0.

[3] $\hat{U}$ is the *decision* utility function, which may have provisions beyond those of the normal utility function. For example, it could penalize configurations that remove resources from an overloaded system to avoid making the system more unstable, even if the normal utility function does not give any value for keeping those resources.

C. Optimizations

We optimized the basic cross-entropy algorithm described in Sec. II-B for our problem domain. Specifically, we added two optimizations to this process:

- **Redistribution to legal actions.** Recall that we use the cross-entropy algorithm (Alg. 1) for each adaptation decision, and each time we ask it to find a *full* adaptation path. We then only execute the first action of this path, and in the subsequent period use Alg. 1 to get another path, because now the initial state is different, given our action in the previous period, and also because the environment realization may have been different than what was predicted. At each state of the system, only a subset of the overall set of actions is feasible. For example, if a system is in a state in which it has the maximum number of servers it supports, then the action "add server" is not valid. Yet since the distributions are calculated based on the history of the sampled solutions (see line 10 in Alg. 1), nothing prevents Alg. 1 from choosing this action.

  As a remedy, as a candidate solution is being generated incrementally, we *redistribute* the current distribution among the actions that are legal after the partial candidate solution generated thus far. Given the current configuration c_0 of the system when the decision is being made, and a partial solution $\sigma = \langle a_1, \ldots, a_k \rangle$, the set of next feasible adaptation actions can be computed as follows. First, we compute the configuration c_k that the system would reach after applying that adaptation path using (4) and (5), taking into account that $\sigma = \langle\rangle \implies c_k = c_0$. Let us denote c_k by $Reach(c_0, \sigma)$. Next, we define the set of valid actions from any configuration c, denoted $ValidAct(c)$, as follows:

  $$ValidAct(c) = \{a : A \mid \exists c' \in C \bullet Act^I(c, c') = a\}$$

  Finally, the set of valid actions after the partial solution σ from configuration c_0 is $ValidAct(Reach(c_0, \sigma))$. Practically, the redistribution of the probability is done as follows.

  1) When the cross-entropy algorithm needs to select randomly action a_i to generate a candidate solution, it gets the set of legal actions after the partial candidate solution $\sigma = \langle a_1, \ldots, a_{i-1} \rangle$ as $ValidAct(Reach(c_0, \sigma))$. Note that the result of $ValidAct$ can be cached, and our implementation does that, to avoid recomputing it multiple times for the same input.
  2) We select a random number $u \in [0, 1]$ and accordingly an action a_i from the CDF (which currently includes illegal actions) that was built based on F_i, as explained before.
  3) If a_i happens to be a legal action, we take it.
  4) Otherwise, we rebuild the CDF as in (1), but based only on the legal actions (i.e., j ranges over the subset of indices in $1..n$ that correspond to legal

actions). This means that $CDF_i(j)$ for the largest j, which we denote by $|CDF_i|$, can be smaller than 1. We thus multiply the random number $u \in [0, 1]$ by $|CDF_i|$ to force it into the legal range, and continue as before.

- **Using hints.** On the one hand, the choice of the next step has to be based on lookahead up to the given horizon H. On the other hand, after each decision we have new information about the environment and hence want to make a new adaptation decision. Since each time the CE algorithm finds a full adaptation path of length H, but we only use the first step of this path, a lot of work seems to be wasted. Our way to (partially) reuse the previous path is as follows. Let σ be the previous path, i.e., the solution produced by the previous invocation of the CE algorithm. The suffix of σ, of length $H - 1$ (all but the first action), serves as a *hint* in the next step.

  However, note that this suffix will be a hint for the *prefix* of length $H - 1$ of the solution to the current decision. This means that, when computing the current solution, for $i = 1 \dots H - 1$, rather than starting with F_i being a uniform distribution, a certain predefined weight is given to the action $\sigma(a_{i+1})$, and the rest is distributed uniformly among the legal actions as determined by the previous optimization.We call this predefined weight the "hint weight" and denote it by ω.

IV. Evaluation

To evaluate the CE-based decision-making approach, we wanted to answer three questions: (i) what speedup PLA-CE attains over PLA-SDP; (ii) how sub-optimal is the solution it computes; and (iii) how effective its decision-making is in practice with respect to PLA-SDP. In addition, we analyze the effect of using the hint optimization on the speedup and the effectiveness of the adaptation decision.

A. Target Example: DART

For the evaluation we used a system that simulates a team of unmanned aerial vehicles (UAVs) developed in the context of the DART (Distributed Adaptive Real-Time) Systems project at the Carnegie Mellon® Software Engineering Institute [46]. The team of drones has a designated leader. They fly with the leader at the center, surrounded by followers in a formation. Two formations are allowed – tight and loose. In the former, the followers stay closer to the leader, while in the latter, they are farther away. High level decisions, such as what formation to adopt, where to fly, or whether to go up or down, are taken autonomously by the leader, and communicated to the rest of the team to be executed.

In particular, we use a scenario of DART where the team has the following mission: to follow a planned route at constant forward speed, detecting as many targets on the ground as possible along the route. Since there are threats along the route that can destroy the team, there is a trade-off between avoiding threats and detecting targets. The environment (i.e., the location of targets and threats) is only discovered during the execution of the mission, and even then, with some uncertainty (due to sensor limitations). Thus, it is not possible to pre-plan the complete execution of the mission. Self-adaptation is required for the team to best deal with the uncertain environment. The team has to adapt by changing altitude and/or formation to maximize the number of targets detected, taking into account that if the formation is lost to a threat, the mission fails. The lower the team flies, the more likely it is to detect targets, but also, the more likely it is to be hit by a threat. Changing formation also involves a similar trade-off, since flying in tight formation reduces the probability of being hit by a threat, but at the same time reduces the chances of detecting targets.

The route is divided into D segments of equal length, and since the team flies at constant speed, there is a direct mapping between time and route segment. The environment state for segment i, referred to as e_i, has two components: the probability that the segment contains a target, and the probability that it contains a threat. These probabilities are obtained by getting multiple observations from forward-looking sensors in the drones. The adaptation goal is to maximize the expected number of targets detected, given by:

$$q = \sum_{t=1}^{D} \left(\prod_{i=1}^{t} s(c_i, e_i) \right) g(c_t, e_t) \tag{9}$$

where $s(c_i, e_i)$ is the probability of survival at time i when the configuration of the team is c_i and the environment is e_i; and $g(c_t, e_t)$ is the probability of detecting a target at time t when the team is in configuration c_t and in environment e_t. The first factor in the summation represents the probability of the team being operational at time t, which requires having survived since the start of the mission. In practice, it is not possible for the self-adaptive system to directly maximize (9), given the limited range of the forward-looking sensors. Therefore, the adaptation goal is approximated by making periodic adaptation decisions—one before entering each route segment—and limiting the sum in (9) to the decision horizon of length H.

B. Speedup

There are two main factors that determine the size of the adaptation-relevant state space in PLA: the configuration state space, and the tactics with latency. The former depends on the number of properties that define the system configuration and how many values each can take. For example, in DART, the more possible altitude levels, the larger the state space. Tactics with latency contribute to the size of the state space because of the need to track their progress. More specifically, for each tactic with latency, there is a state variable that indicates how many periods are left until the tactic completes. To assess the speedup that PLA-CE attains over PLA-SDP, we ran experiments with 22 different combinations of features and tactics that resulted in different state space sizes. For each of these combinations, we ran 30 simulations of missions with a route length of 75 segments, for a total of 2250 adaptation decisions. Each simulation was executed with PLA-SDP as

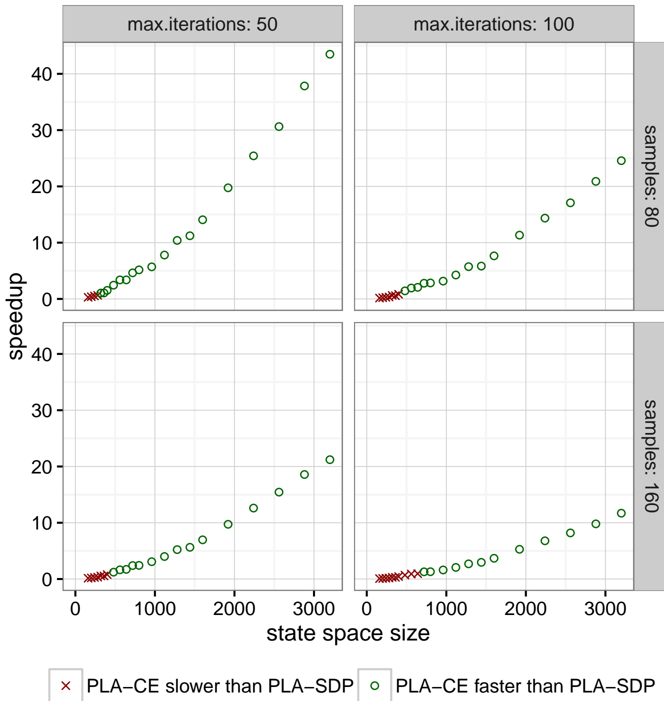

Fig. 1. PLA-CE speedup over PLA-SDP for DART.

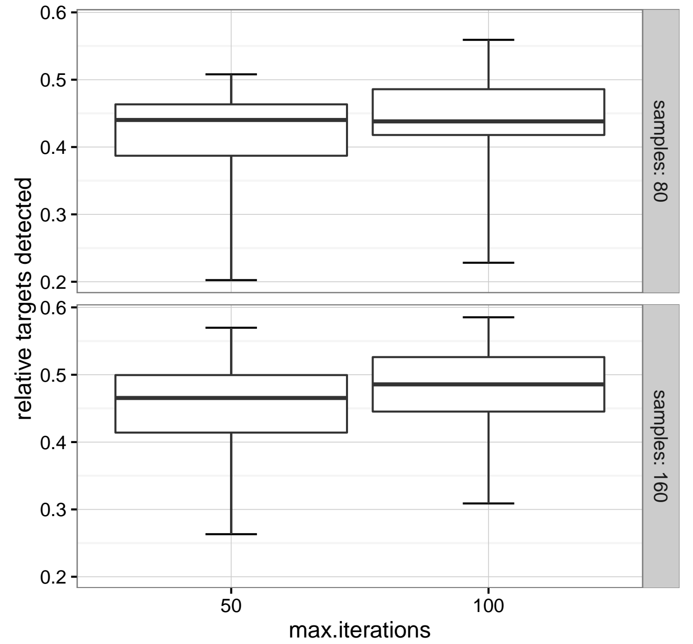

Fig. 2. PLA-CE solution effectiveness relative to optimal solution for DART.

the baseline for comparison, and with PLA-CE using different combinations of sample sizes and maximum iterations. In all cases, the other parameters of PLA-CE were set as follows: $\rho = 0.1$, $\epsilon = 0.01$, $\alpha = 0.2$, and $\omega = 0.3$.[4]

Fig. 1 shows the speedup (i.e., how many times faster) PLA-CE was compared to PLA-SDP. The results show that for small state spaces, PLA-CE is slower. However, the speedup grows exponentially with the state space size, and becomes faster even for state spaces of modest size. As expected, the smaller the maximum number of iterations and the number of samples, the higher the speedup, going from 10 times up to more than 40 times for the case with the largest state space.

C. *Effectiveness Relative to Optimal*

PLA-CE makes adaptation decisions much faster than PLA-SDP, but it finds a solution that approximates the optimal. Thus, it is interesting to quantify how much worse than the optimal solution the CE-based solution is. The solution that PLA-SDP computes is optimal in expectation, meaning that it maximizes the *expected* utility over the decision horizon given the probabilistic behavior of the environment and the sensors. This solution, however, is not necessarily optimal a posteriori, once the realization of the environment and the sensors is known. Furthermore, since the decision is optimal in expectation only over the decision horizon, it may not be optimal when the behavior of the environment beyond the horizon is factored in, something that is not possible to do in practice given the range of the sensors.

In order to compare against a solution that is optimal a posteriori, we removed all the random behaviors that happen during the simulation, such as the behavior of sensors and threats, and configured the forward-looking sensors to have no error and infinite range. The initial placement of threats and targets was still random, but determined by a seed, so that each case could be replicated with the different solution approaches. In addition, instead of computing the adaptation decision periodically, the system was modified to compute the complete adaptation path at the beginning of the mission, something that was possible because the sensors had unlimited range and no error. With these modifications, PLA-SDP computes the a posteriori optimal solution. Note that for these experiments PLA-CE was run with the same modifications, thus computing an approximation to the a posteriori optimal solution.

Each box plot in Fig. 2 summarizes the statistics of the number of targets detected by PLA-CE relative to the optimal solution in 600 simulated missions. The bottom and top of the box represent the 1st and 3rd quartiles respectively, the line in between is the median, and the whiskers represent the range. The results show that in the vast majority of cases, PLA-CE finds less than 50% of the targets the optimal solution finds, even with a large sample size and a large number of iterations.

D. *Effectiveness in Practice*

Given the limitations of the sensors used to estimate the environment ahead, and the stochastic behavior of the environment, it is not possible to make optimal a posteriori adaptation decisions. With these limitations, in practice, adaptation

[4]We did not fine-tune these parameters in order to avoid over-fitting for a particular system. In early experiments, we noticed that $\alpha = 0.2$ performed in general slightly better than values $0.1, 0.3, 0.4$, but did not dominate them. We also tried $\epsilon = 0.001$ but decided to use $\epsilon = 0.01$ to achieve faster convergence. The value for ω was chosen from the results shown in Figure 4. Note that we used the same parameter values for the two very different systems used in the evaluation.

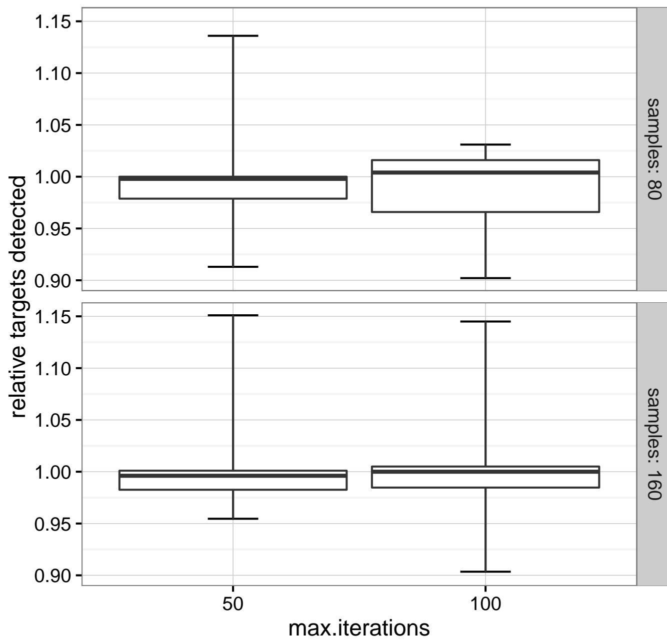

Fig. 3. PLA-CE solution effectiveness relative to PLA-SDP for DART.

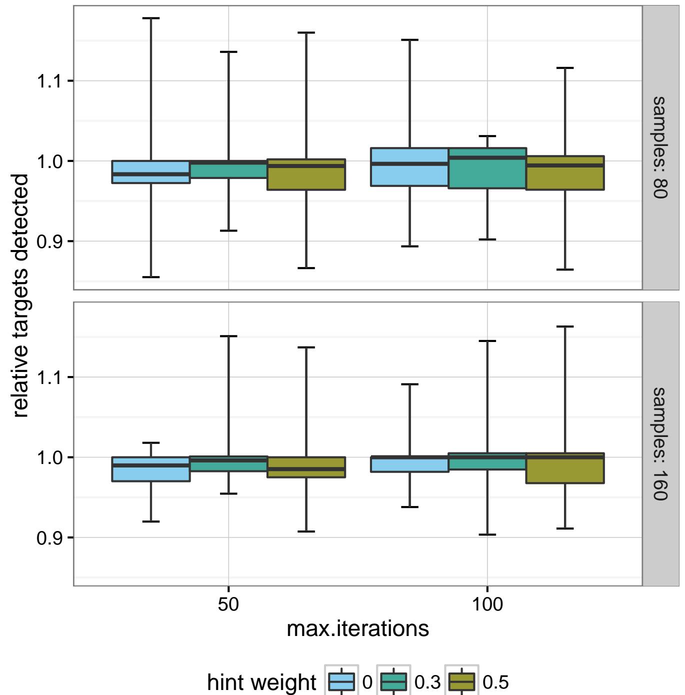

Fig. 4. Effect of hint weight value on solution effectiveness relative to PLA-SDP for DART.

decisions can at best be optimal in expectation, and over a limited decision horizon, as is the case with PLA-SDP.

To evaluate the effectiveness of PLA-CE in practice, we repeated the previous experiments, but this time using the full simulation with all its random effects. Fig. 3 shows that in practice the effectiveness of PLA-CE is very close to that of PLA-SDP, even for low iterations and sample size, which can be up to 40 times faster as was previously shown. Furthermore, in some cases PLA-CE performed better than PLA-SDP, as shown by the outliers, and sometimes the 3rd quartile, that lay above 1. The reason for this is that with all the stochastic behavior, a decision that is optimal in expectation may perform suboptimally under some realizations of the environment (e.g., in an environment state that had low probability, and consequently, low weight in the selection of the solution). Conversely, a decision that is suboptimal in expectation may yield better results than the optimal decision in expectation for some realizations of the environment (e.g., a solution that happened to be better for a realization that had low probability).

E. Hint Effect

Intuitively, the hint optimization bootstraps the adaptation decision by giving more weight in the initial sampling distribution to a likely prefix of the best adaptation path. This can potentially result in faster convergence to the solution, which, in turn can speed up the decision, and in cases in which it does not converge because it reaches the iteration limit, the resulting solution is likely to be better if the hint biased the sampling towards the optimal solution. To evaluate whether this is actually the case, we repeated the experiments with different hint weights, and with no hint (i.e., $\omega = 0$).

Figure 4 shows the results, comparing the effectiveness of PLA-CE with different hint weights relative to PLA-SDP. We can observe that $\omega = 0.3$ is better than $\omega = 0$, if we consider that the median of the former is higher. If we also consider the minimum (lower whisker end) and the 1st quartile, we observe that the hint is more beneficial when we have a lower limit on the number of iterations. This matches our conjecture that the hint allows reaching a better solution within the iterations allowed. If a larger number of iterations is allowed, the CE algorithm is more likely to converge even if no hint is provided, which is reflected in the lesser improvement observed when the maximum number of iterations is 100. We can also observe in Figure 4 that giving the hint too much weight is detrimental, since it is worse than not using a hint at all.

Figure 5 shows how the hint affects the decision time. When the number of iterations is limited to 50, the decision time improvement is negligible. This is because CE is still using all the iterations it is allowed to in most cases, with the hint allowing it to reach a better solution as shown in Figure 4. However, when the limit of iterations is 100, the improvement in speed is noticeable. In this case, the hint is allowing CE to converge faster. However, when looking at these results combined with the effect of the hint on quality, we observe that when the hint weight is too high, CE appears to be converging faster to a non-optimal solution. In conclusion, the use of the hint appears to be more useful when the running time of CE is limited the most.

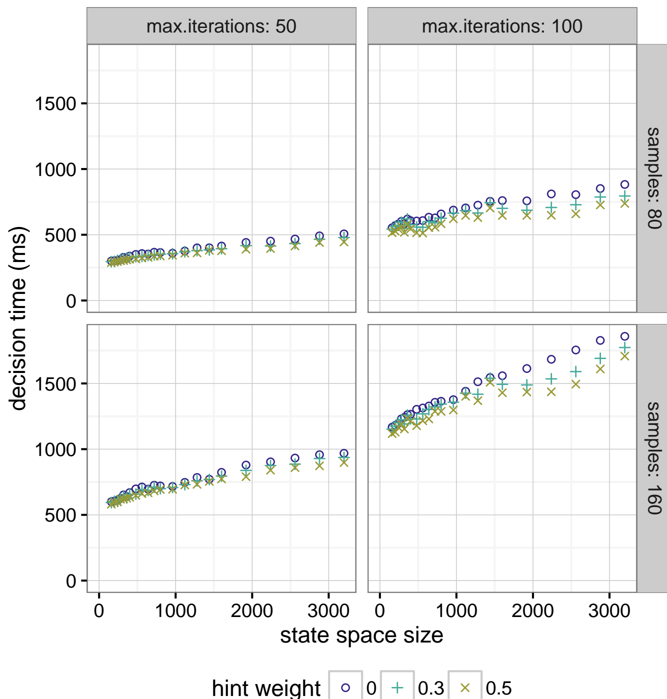

Fig. 5. Effect of hint on the decision time of PLA-CE for DART.

F. Experiments with a Web Application

In addition to measuring the effectiveness with DART, we ran a smaller experiment using a completely different system that comes from a different domain, and differs also in its adaptation goal and the repertoire of tactics used. In this case, we used SWIM, a simulation of web applications with a 3-tier architecture similar to other systems used in self-adaptive systems research such as RUBiS [47] and Znn [48]. SWIM simulates a web application with a load balancer that distributes requests sent by users among a number of servers that can process these request. SWIM does not simulate the functionality of the web application. Instead, the processing of requests is simulated simply as a computation that takes time, which is simulated by computing the time it would take to service the request sharing the server's processor, as is done in a web server. SWIM supports the brownout paradigm, which allows controlling the load induced by the traffic to the website by controlling the proportion of responses that include optional content, such as related items [49]. The traffic to the website is simulated by replaying access traces recorded from real websites. For these experiments, a trace from the WorldCup '98 trace archive [50] was used.

The simulation provides effectors to add and remove servers at run time, and to increase and decrease the dimmer value, the setting that controls the proportion of responses for which the optional content is computed and included. The adaptation manager uses adaptation tactics that map directly to these four effectors. In addition, SWIM provides monitoring probes that the adaptation manager can use to monitor the state of the system and the environment. The state space size is determined by the maximum number of servers supported by the system, and by the number of levels in which the continuous range for the dimmer setting is discretized. The tactic to add a server has latency, and its state also contributes to the state space size.

TABLE I
PLA-CE SPEEDUP AND EFFECTIVENESS RELATIVE TO PLA-SDP IN SIMULATION OF WEB SYSTEM.

state space	servers	dimmer levels	speedup	relative effectiveness
small	3	5	0.1	0.99
medium	30	10	7.9	1.08
large	30	20	29.3	0.94

The goal of self-adaptation in this system is to maximize the utility accrued while minimizing cost. The utility consists of positive rewards accrued minus penalties, which are defined by a service level agreement (SLA). The rewards are gained by meeting the average response time requirement, whereas penalties are incurred when the requirement is not satisfied. The cost is the number of servers used.

The experiments were run configuring the system with three different combinations of number of servers and dimmer levels to obtain three different state spaces sizes, as shown in the first three columns on Table I. For each, the simulation was run with both PLA-SDP and PLA-CE. The parameters for the latter were $N = 50$, $\rho = 0.1$, $\epsilon = 0.01$, $\alpha = 0.2$, and $\omega = 0.3$, with the number of iterations limited to 75. Since all the random behavior in the simulation is controlled by a seeded random number generator, the exact same conditions were replicated for both approaches. Table I shows results similar to those obtained in the experiments with DART. When the state space is small, PLA-CE is slower than PLA-SDP, but for lager state spaces, PLA-CE is faster, close to 30 times for the largest in these runs. Also, the effectiveness of PLA-CE is very close to that of PLA-SDP, regardless of the state space size. We conjecture that for large state spaces due to finer grained discretization of system properties, such as the dimmer setting, if the approximate solution is off by one or two levels from the optimal, it probably does not affect the outcome in a significant way. On the other hand, for coarse grained discretization, as in the small state space, being off by one level could have a substantial impact. However, in this case, it is likely that CE will converge to the optimal solution within the allotted number of samples and iterations, given that the state space is small.

V. Related Work

Runtime quantitative verification (RQV) is an approach to self-adaptation that uses probabilistic model checking at run time to predict violations of requirements, and in that case, select, for instance, the configuration less likely to result in an unsatisfied requirement at the lowest cost, which is also done using model checking [3]. One issue with this approach is that the use of a model checker at run time can impose an unaccept-

able overhead for adaptation decisions in some systems. For these reason, researchers have worked on speeding up RQV. Gerasimou et al. propose a suite of techniques that speedup RQV making it practical for self-adaptation [10]. In particular, they show how the overhead and execution time of RQV can be reduced by combining caching of previously computed solutions, lookahead to compute in advance solutions that may be needed, and near-optimal reconfiguration to stop the search for the best solution when one good enough has been found. The latter technique is the one that most resembles our approach given that both may settle on an approximation to the optimal solution. However, their approach does not use cross-entropy to narrow the solution space progressively towards the optimal. Our use of a hint to bootstrap decisions is somewhat similar to their caching and look-ahead, in that the hint is a suffix of a solution previously computed, and represents an adaptation path computed over the decision look-ahead horizon. However, in their look-ahead, solutions are computed outside of the periodic decision using spare CPU cycles, whereas our hint is a byproduct of the previous adaptation decision. Therefore, the hint is available with our approach even if there are no idle CPU cycles between adaptation decisions.

Filieri et al. also propose work to speed up the use of probabilistic model checking at run time [51]. Their work focuses on the verification of probabilistic properties in R-DTMCs (i.e., DTMCs extended with rewards) where some transitions have probabilities and rewards that are updated at run time. Their approach pre-computes, at design time, expressions that can be quickly evaluated at run time to determine the satisfaction of the system's requirements. Our work also performs part of the computation off-line, something which is inherited from PLA-SDP. In addition, we allow the run-time update of transition probabilities (for the environment part of the model), and values associated with environment states (e.g., the request arrival rate at a website) and rewards given by the utility function driving adaptation decisions. Other than these similarities, the approaches are different in that theirs supports the verification of general probabilistic properties extended with rewards, whereas ours, based on MDPs, makes decisions selecting actions that maximize the expected reward.

The CE method has been used to find policies for MDPs in other approaches [52], [53], [54]. The closest to ours is the approach for planning in large-scale stochastic domains presented by Wienstein and Littman [53]. Their approach also applies the CE method to find policies for MDPs using a finite receding horizon and committing to the first action in the policy. In their approach, all sequences of actions are feasible. However, in self-adaptation, not all sequences of adaptation actions are feasible, and our approach handles that. In addition, their approach does not consider the latency of the different actions as ours does. A third difference is that their approach does not leverage previously computed policies as we do with our hint.

VI. Conclusion

We have presented PLA-CE, an approach to make proactive latency-aware adaptation decisions using the cross-entropy method for optimization. PLA-CE finds the solution to the adaptation decision problem by randomly sampling from the solution space, adjusting the sampling distribution in each iteration so that the probability of finding the optimal solution increases. Even though the CE method does not guarantee convergence to the optimal solution, we found that in most cases, it achieves an effectiveness close to 99% of that of PLA-SDP, but does it up to 40 times faster. Furthermore, similar outcomes were obtained with two very different self-adaptive systems that have different tactic repertoires and adaptation goals.

Although in this paper we have used the CE method to make PLA adaptation decisions faster, we conjecture that the same approach could be used to speed up other complex self-adaptation decisions whose running time would be excessive for some systems. In future work, we plan on developing a distributed version of PLA-CE, parallelizing the sampling and evaluation of solutions in the CE algorithm to leverage computing capacity in other processors, such as the on-board computers of the drones in the same team.

Acknowledgment

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. [Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution. Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University. (DM-0004347)

References

[1] B. H. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle, "Software Engineering for Self-Adaptive Systems: A Research Roadmap," in *Software Engineering for Self-Adaptive Systems*, ser. Lecture Notes in Computer Science, B. H. C. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, jun 2009, vol. 5525, pp. 1–26.

[2] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and Research Challenges," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 4, no. 2, pp. 1–42, may 2009.

[3] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola, "Self-adaptive software needs quantitative verification at runtime," *Communications of the ACM*, vol. 55, no. 9, p. 69, sep 2012.

[4] A. Naskos, E. Stachtiari, P. Katsaros, and A. Gounaris, "Probabilistic Model Checking at Runtime for the Provisioning of Cloud Resources," *Runtime Verification*, 2015.

[5] N. Esfahani, E. Kouroshfar, and S. Malek, "Taming uncertainty in self-adaptive software," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. New York, NY, USA: ACM, 2011, pp. 234–244.

[6] J. Cámara, A. Lopes, D. Garlan, and B. Schmerl, "Impact Models for Architecture-Based Self-Adaptive Systems," in *Proceedings of the 11th International Symposium on Formal Aspects of Component Software (FACS2014)*, Bertinoro, Italy, 2014.

[7] J. Cámara, D. Garlan, B. Schmerl, and A. Pandey, "Optimal planning for architecture-based self-adaptation via model checking of stochastic games," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, ser. SAC '15. New York, NY, USA: ACM, 2015, pp. 428–435.

[8] S. Iannucci and S. Abdelwahed, "A probabilistic approach to autonomic security management," in *2016 IEEE International Conference on Autonomic Computing (ICAC)*, July 2016, pp. 157–166.

[9] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl, "Proactive self-adaptation under uncertainty: a probabilistic model checking approach," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*. New York, New York, USA: ACM Press, aug 2015, pp. 1–12.

[10] S. Gerasimou, R. Calinescu, and A. Banks, "Efficient runtime quantitative verification using caching, lookahead, and nearly-optimal reconfiguration," in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems - SEAMS 2014*. New York, New York, USA: ACM, jun 2014, pp. 115–124.

[11] G. A. Moreno, J. Camara, D. Garlan, and B. Schmerl, "Efficient Decision-Making under Uncertainty for Proactive Self-Adaptation," in *2016 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, jul 2016, pp. 147–156.

[12] R. Rubinstein, "The cross-entropy method for combinatorial and continuous optimization," *Methodology and computing in applied probability*, vol. 1, no. 2, pp. 127–190, 1999.

[13] M. Mao and M. Humphrey, "A Performance Study on the {VM} Startup Time in the Cloud," in *2012 IEEE Fifth International Conference on Cloud Computing*. IEEE, jun 2012, pp. 423–430.

[14] M. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, Ltd, 2014.

[15] Z. Botev, D. Kroese, R. Rubinstein, and P. L'Ecuyer, "The cross-entropy method for optimization," in *Machine Learning*, ser. Handbook of Statistics, V. Govindaraju and C. Rao, Eds. Elsevier, 2011, vol. 31.

[16] R. Y. Rubinstein, "Optimization of computer simulation models with rare events," *European Journal of Operational Research*, vol. 99, no. 1, pp. 89–112, 1997.

[17] G. Rubino and B. Tuffin, *Rare Event Simulation using Monte Carlo Methods*. Wiley, 2009.

[18] R. Y. Rubinstein, "The cross-entropy method for combinatorial and continuous optimization," *Methodology and Computing in Applied Probability*, vol. 1, no. 2, pp. 127–190, 1999.

[19] ——, "Combinatorial optimization, cross-entropy, ants and rare events," in *Stochastic Optimization: Algorithms and Applications*, S. Uryasev and P. M. Pardalos, Eds. Dordrecht: Kluwer, 2001, pp. 304–358.

[20] L. Margolin, "On the convergence of the cross-entropy method," *Annals of Operations Research*, vol. 134, no. 1, pp. 201–214, 2005.

[21] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Statist.*, vol. 22, no. 1, pp. 79–86, 03 1951.

[22] R. P. Kothari and D. P. Kroese, "Optimal generation expansion planning via the cross-entropy method," in *Proceedings of the 41st conference on Winter simulation*, ser. WSC '09. Winter Simulation Conference, 2009, pp. 1482–1491.

[23] A. Sani, "Stochastic modelling and intervention of the spread of HIV/AIDS," Ph.D. dissertation, The University of Queensland, Brisbane, 2009.

[24] A. Sani and D. P. Kroese, "Controlling the number of HIV infectives in a mobile population," *Mathematical Biosciences*, vol. 213, no. 2, pp. 103–112, 2008.

[25] D. P. Kroese, S. Porotsky, and R. Y. Rubinstein, "The cross-entropy method for continuous multi-extremal optimization," *Methodology and Computing in Applied Probability*, vol. 8, no. 3, pp. 383–407, 2006.

[26] Z. Szabó, B. Póczos, and A. Lörinc, "Cross-entropy optimization for independent process analysis," *In Independent Component Analysis and Blind Signal Separation*, vol. 3889, pp. 909–916, Springer-Verlag, Heidelberg, 2006.

[27] L. Busoniu, R. Babuska, B. D. Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming using Function Approximators*. New York: Taylor & Francis Group, 2010.

[28] Z. I. Botev and D. P. Kroese, "Global likelihood optimization via the cross-entropy method with an application to mixture models," in *Proceedings of the 36th conference on Winter simulation*, ser. WSC '04. Winter Simulation Conference, 2004, pp. 529–535.

[29] A. Boubezoula, S. Paris, and M. Ouladsinea, "Application of the cross entropy method to the GLVQ algorithm," *Pattern Recognition*, vol. 41, no. 10, pp. 3173–3178, 2008.

[30] D. P. Kroese, R. Y. Rubinstein, and T. Taimre, "Application of the cross-entropy method to clustering and vector quantization," *Journal of Global Optimization*, vol. 37, pp. 137–157, 2007.

[31] Z. Liu, A. Doucet, and S. S. Singh, "The cross-entropy method for blind multiuser detection," in *Proceedings International Symposium on Information Theory*, Piscataway, Chicago, 2004.

[32] J. Keith and D. P. Kroese, "Rare event simulation and combinatorial optimization using cross entropy: sequence alignment by rare event simulation," in *Proceedings of the 34th conference on Winter simulation*, ser. WSC '02. Winter Simulation Conference, 2002, pp. 320–327.

[33] V. Pihur, S. Datta, and S. Datta, "Weighted rank aggregation of cluster validation measures: a Monte Carlo cross-entropy approach," *Bioinformatics*, vol. 23, no. 13, pp. 1607–1615, 2007.

[34] J.-C. Chen, C.-K. Wen, C.-P. Li, and P. Ting, "Cross-entropy optimization for the design of fiber Bragg gratings," *Photonics Journal, IEEE*, vol. 4, no. 5, pp. 1495 –1503, oct. 2012.

[35] G. Alon, D. Kroese, T. Raviv, and R. Rubinstein, "Application of the cross-entropy method to the buffer allocation problem in a simulation-based environment," *Annals of Operations Research*, vol. 134, no. 1, pp. 137–151, 2005.

[36] I. Cohen, B. Golany, and A. Shtub, "Resource allocation in stochastic, finite-capacity, multi-project systems through the cross entropy methodology," *Journal of Scheduling*, vol. 10, no. 1, pp. 181–193, 2007.

[37] D. P. Kroese, K.-P. Hui, and S. Nariai, "Network reliability optimization via the cross-entropy method," *IEEE Transactions on Reliability*, vol. 56, no. 2, pp. 275–287, 2007.

[38] K. Chepuri and T. Homem-de-Mello, "Solving the vehicle routing problem with stochastic demands using the cross entropy method," *Annals of Operations Research*, vol. 134, no. 1, pp. 153–181, 2005.

[39] D. Ernst, M. Glavic, G.-B. Stan, S. Mannor, and L. Wehenkel, "The cross-entropy method for power system combinatorial optimization problems," in *Proceedings of the 7th IEEE Power Engineering Society (IEEE-PowerTech 2007)*, 2007, pp. 1290–1295.

[40] A. Lörincza, Z. Palotaia, and G. Szirtesb, "Spike-based cross-entropy method for reconstruction," *Neurocomputing*, vol. 71, no. 16-18, pp. 3635–3639, 2008.

[41] I. Menache and S. Mannor, "Basis function adaptation in temporal difference reinforcement learning," *Annals of Operations Research*, vol. 134, no. 1, pp. 215–238, 2005.

[42] A. Ünveren and A. Acan, "Multi-objective optimization with cross entropy method: Stochastic learning with clustered pareto fronts," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2007, pp. 3065–3071.

[43] Y. Wu and C. Fyfe, "Topology perserving mappings using cross entropy adaptation," in *Proceedings of the 7th WSEAS International Conference on Artificial intelligence, knowledge engineering and data bases*, ser. AIKED'08. Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS), 2008, pp. 176–181.

[44] G.-B. Chaslot, M. Winands, I. Szita, and H. van den Herik, "Cross-entropy for Monte-Carlo tree search," *ICGA Journal*, vol. 31, no. 3, pp. 145–156, 2008.

[45] R. Y. Rubinstein, A. Ridder, and R. Vaisman, *Fast Sequential Monte Carlo Methods for Counting and Optimization*. New York: John Wiley & Sons, 2013.

[46] S. A. Hissam, S. Chaki, and G. A. Moreno, "High Assurance for Distributed Cyber Physical Systems," in *Proceedings of the 2015 European Conference on Software Architecture Workshops*. New York, New York, USA: ACM Press, sep 2015, pp. 1–4.

[47] "RUBiS: Rice University Bidding System," http://rubis.ow2.org/.

[48] S.-W. Cheng, D. Garlan, and B. Schmerl, "Evaluating the effectiveness of the Rainbow self-adaptive system," in *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, may 2009, pp. 132–141.

[49] C. Klein, M. Maggio, K.-E. Årzén, and F. Hernández-Rodriguez, "Brownout: building more robust cloud applications," in *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*. New York, New York, USA: ACM, may 2014, pp. 700–711.

[50] M. Arlitt and T. Jin, "A workload characterization study of the 1998 World Cup web site," *IEEE Network*, vol. 14, no. 3, pp. 30–37, 2000.

[51] A. Filieri, G. Tamburrelli, and C. Ghezzi, "Supporting self-adaptation via quantitative verification and sensitivity analysis at run time," *IEEE Transactions on Software Engineering*, vol. 42, no. 1, pp. 75–99, Jan 2016.

[52] P. T. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of Operations Research*, vol. 134, no. 1, pp. 19–67, 2005.

[53] A. Weinstein and M. L. Littman, "Open-loop planning in large-scale stochastic domains," in *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, ser. AAAI'13. AAAI Press, 2013, pp. 1436–1442.

[54] F. Celeste, F. Dambreville, and J. p. Le Cadre, "Optimal path planning using cross-entropy method," in *2006 9th International Conference on Information Fusion*, July 2006, pp. 1–8.