SASS: Self-adaptation using stochastic search

Zack Coker, David Garlan, Claire Le Goues
School of Computer Science
Carnegie Mellon University, Pittsburgh, PA 15213
Email: {zfc,dgarlan,clegoues}@cs.cmu.edu

***Abstract*—Future-generation self-adaptive systems will need to be able to optimize for multiple interrelated, difficult-to-measure, and evolving quality properties. To navigate this complex search space, current self-adaptive planning techniques need to be improved. In this position paper, we argue that the research community should more directly pursue the application of *stochastic search techniques*—search techniques, such as hill climbing or genetic algorithms, that incorporate an element of randomness—to self-adaptive systems research. These techniques are well-suited to handling multi-dimensional search spaces and complex problems, situations which arise often for self-adaptive systems. We believe that recent advances in both fields make this a particularly promising research trajectory. We demonstrate one way to apply some of these advances in a search-based planning prototype technique to illustrate both the feasibility and the potential of the proposed research. This strategy informs a number of potentially interesting research directions and problems. In the long term, this general technique could enable sophisticated plan generation techniques that improve domain specific knowledge, decrease human effort, and increase the application of self-adaptive systems.**

I. Introduction

True software self-adaptation requires the efficient traversal of an expansive and multi-dimensional problem space. Future-generation systems will need to be able to optimize for multiple, interacting, difficult-to-measure, and evolving self-* properties and priorities. While promising, existing solutions tend to consider only one or two properties at a time, assume property independence, fail to consider properties with evolving priorities, or more generally are insufficiently complex to meet the needs of the next generation of self-adaptive software systems. We need more sophisticated solutions to profitably navigate this complex, multi-objective search space.

We believe that a promising research avenue towards such sophisticated solutions lies in stochastic search-based software engineering techniques. Stochastic search techniques are search mechanisms that incorporate an element of randomness; well-known examples include hill climbing and genetic algorithms. Search-based software engineering applies such approaches to a diversity of problems in software engineering [1]. Although there exists promising interest in this area (e.g., [2]–[4]), including a keynote on the subject at SEAMS 2014 [5], and a recently-instated SBSE-SS (for Self-search) track at GECCO [6], there is strikingly little previous work in directly applying such techniques to software self-adaptation, or to software architecture design or evolution in general.

In this position paper, we argue that the time is right to address this research oversight. We believe that applying search based software engineering strategies to problems in multi- and evolving-objective self-adaptive software systems is both *promising* and *possible*. We believe that it is *promising* because stochastic search-based techniques are particularly well-suited to addressing problems of this form, and because we believe that the combination of these types of techniques will provide unique insights about how to design and implement self-* systems. We believe it is *possible* because of recent advances that admit the efficient, automatic representation and evaluation of key problems in self-adaptive systems. We substantiate these claims with a proof-of-concept prototype search-based planner, applied to the well-known Znn.com example. This proof of concept demonstrates one way that stochastic techniques can be applied to contingency planning for nondeterministic system behavior, plan improvement, changing utility conditions, multiple interdependent utility conditions, and plans involving complex actions that take time and may fail.

Although we focus primarily on the planning aspect of the self-adaptation loop for the purposes of a concrete discussion, we believe that the space of potential research approaches for SBSE to self-adaptation is quite broad. The design space of self-* software systems inherently involves trade-offs in multiple interrelated and evolving dimensions, including complex notions of time, cost, and non-determinism. We therefore believe that this initial set of ideas holds promise at multiple levels in the space of self-* software systems research.

The rest of this paper proceeds as follows. We begin with motivation, focusing on what is missing from existing knowledge on self-adaptive systems that prevents their full realization as well as the features of the problem space that lend it particularly well to stochastic, SBSE-inspired search techniques (Section II). We then provide background on stochastic search and search-based software engineering to substantiate our claims that the approaches are particularly promising in this space (Section III). We use a prototype search-based planner to illustrate potentially interesting directions in this research area in Section IV. We discuss potential directions for the proposed research program in Section V, and then conclude (Section VI).

II. The search space of self-* systems

The ideal self-adaptive system is general and flexible, able to optimize for multiple and difficult-to-measure quality properties and respond to changing priorities and evolving constraints. Unfortunately, despite high-level research success in generally characterizing the problem space (including the

MAPE-loop conception of self-* software systems [7] as well as more recent treatments [8]), repeated surveys of the field have found that most existing techniques commonly address only one or two static or independent properties [9], [10].

This approach is inadequate for the next generation of self-adaptive systems. Consider the planning component of the MAPE loop. The ideal self-adaptive planner must be able to adjust to a wide variety of situations. It must be able to adjust from single systems with thousands of configurations, where deterministically searching the configuration search space becomes infeasible [11], to Ultra Large Scale systems, where multiple systems are distributed with complex interactions and different goals [12]. Planners also need to be able to adjust to specific run-time contexts, reconfigurations, new adaptation, and changing system policies [13]. Of course, many planning advances in autonomous systems have been successfully adapted to software [3], [14], [15]. However, while there have been investigations into, for example, goal-oriented plans for self-adaptive systems [16], [17], the complexity of software and environmental interactions have typically prevented previous techniques from applying to many situations [18]. Even adaptable plans [11] are largely unable to adapt beyond specific situations, and are better suited for failure recovery, as compared to optimizing a quality at run-time [19]. Run-time feedback decision-making may help improve system reliability [13], but existing work still tends to only provide guarantees for restricted forms of system goals, or only in specific systems (e.g., [20]), rather than across the full span of possible self-* properties.

These complexities extend beyond those inherent to the existence of multiple quality attributes of interest. Interactions between competing objectives need to be investigated further [10], and objective priorities may change over a system's lifetime. Beyond this, self-adaptation actions take *time*, and can *fail*. Previous research has proposed the *automatic workarounds* approach [21], [22], which consists of using redundant or similar features to bypass problematic application features. As self-adaptive systems become more complex, multiple different adaptation sequences can be taken to produce similar results for some measurable quality. However, the interactions between failure handling actions and other quality attributes of interest (adaptation sequence cost, for example) can be non-trivial. While these sequences are similar for one quality, they may have vastly different and unknown effects on the whole system of qualities that a designer cares about.

III. Stochastic search and SBSE

Stochastic search techniques use randomness to avoid some of the limitations and biases of deterministic heuristics. Instead, stochastic techniques only need to be able to evaluate a given solution, and reduce the space by efficiently directing the search toward the higher scoring solutions using well-chosen representation and operators.

In this paper, we argue that the research community should strive to fully adapt stochastic search methods from search-based software engineering (SBSE) to the problem of software self-adaptation. Search-based software engineering applies stochastic search methods to a diversity of problems in software engineering [1], including testing [23]–[25], development process and effort estimation [26], and program evolution and repair [27], [28]. Although there has been previous success and interest in applying such methods to self-* software systems and architectural design or evolution more generally (e.g., [2], [3]), the amount of previous work in this space is remarkably low. Harman's recent comprehensive survey [29], for example, does not include self-adaptive software in its taxonomy of the SBSE field. However, the survey does note several "overlooked and emerging areas" of direct interest, including on-line optimization, SBSE for non-functional properties, and multi-objective optimization.

At a high level, a stochastic search technique traverses a space of candidate solutions to a problem, which are encoded in an underlying *representation*. Each candidate is assigned a score based on how close it is to the desired solution, using an *objective* or *fitness function.* Typically, solutions that are closer to the goal (have higher fitness) are more likely to be retained to contribute to the search in future iterations. New candidates are created by applying domain-specific *operators* to previous solutions. Critically, by definition, this process includes an element of randomness, such as in choosing which operators to apply or which candidates to retain. Examples of well-known stochastic search methods in SBSE include hill climbing and genetic algorithms [30].

We believe that these techniques are particularly promising for self-adaptive software systems, for several reasons. Recent work demonstrated the potential of SBSE techniques to solve problems at real-world scale, such as bug repair in large, extant software systems [27], [28]. This has resulted in a body of knowledge on how to scalably manipulate tree-based program constructs in terms of both representation and operator encodings (such as representing candidate program improvements as sets of changes rather than entire solutions [31]). Stochastic techniques have also recently been shown to successfully apply past knowledge to new situations [32], [33], suggesting promise in helping long-running systems evolve in response to new quality objectives or priorities. Stochastic search-based techniques have been useful in developing unusual or unexpected solutions to difficult problems. The results of such techniques may therefore be informative for researchers trying to understand the design space of self-adaptive systems.

Finally, although predicting and measuring hard-to-quantify quality attributes is an open problem, recent advances in probabilistic model checking (see, e.g., PRISM [34]) allows us to quantify expected costs and benefits in a given self-adaptive system or associated modification plan. This is important because it suggests a way to evaluate candidate solutions in unsupervised search. Such tools are especially interesting in this domain because they provide continuous (probabilistic) measures of the quality of a candidate solution. Multi-objective search problems remain challenging, but we believe such tools provide a promising avenue to evaluating candidate fitness, which we demonstrate with our prototype tool, discussed next.

Basic Action	Precondition	Cost ($)	Resp. Time	Time	Failure Rate
Add S_1 Server	$S_1 < S_1^{max}$	15	-5	600	0.1
Add S_2 Server	$S_2 < S_2^{max}$	20	-5	600	0.1
Remove S_1 Server	$S_1 > 1$	-15	5	600	0.1
Remove S_2 Server	$S_2 > 1$	-20	5	600	0.1
Add DB_A Thread	$T_A < T_A^{max}$	0	-2	180	0.2
Add 2 DB_B Threads	$T_B < T_B^{max}$-1	0	-1	180	0.2
Increase Quality	Quality set low	0	-2S	1	0.3
Decrease Quality	Quality set high	0	2S	60	0.3

TABLE I

BASIC ACTIONS: S_1 and S_2 refer to the server pools at locations 1 and 2 respective; S is the total number of active servers. T_A and T_B refers to the number of threads associated with databases A and B, respectively; T_N^{max} is the maximum permitted number of threads in each database, which we set to 5 in our our simulations. Each action is associated with a precondition, which governs whether it may be applied; an adaptation cost (which is incurred whether the system succeeds or not); and an effect on system response time. Each action takes some amount of Time to apply (regardless of success or failure), and may fail, with a probability shown in Failure Rate.

$$
\begin{aligned}
G \quad ::= \quad & \text{basicAction} \\
\mid \quad & \text{if } (G) \text{ succeeds then } (G) \text{ else } (G) \\
\mid \quad & G \ ; \ G
\end{aligned}
$$

Fig. 1. Compound actions consist of either basic actions or control actions. `ifSuccessElse` allows plan branching. If the first action executes successfully, the plan executes the basic action associated with the success branch; otherwise, it executes the action on the failure branch. `Seq` (indicated via the semicolon in example plans), indicates two actions to be executed in sequence. The second action is always attempted regardless of the outcome of the first.

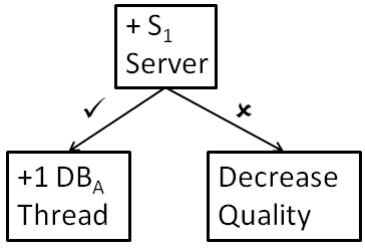

Fig. 2. Example plan: This plan states that the system should first try to add a server at location 1. If that adaptation is successful, the system should increase the thread count for database A. If adding a server at location 1 fails, then the system should reduce the resolution of the system. This plan is feasible from the current initial configuration.

IV. PROOF OF CONCEPT

In this section, we describe a prototype that uses *genetic programming* (GP) to perform automated, self-adaptive planning in a system with multiple competing, time sensitive, probabilistic quality objectives. GP is the application of genetic algorithms (GA) to search problems involving tree-based solutions (typically programs); a GA is a population-based, iterative stochastic search method inspired by biological evolution [30].[1] GP uses computational analogues of biological mutation and crossover to generate new candidate solutions, and evaluates solutions using a domain-specific objective, or *fitness* function. Potential solutions that have a high fitness score are more likely to be randomly retained into future iterations, both alone, modified slightly (via mutation), or in combination with other potentially partial solutions. We demonstrate the use of a probabilistic model checker as a fitness function, enabling the exploration of a complex multi-dimensional search space with evolving, conflicting, and interacting quality objectives. The current version of the planner creates plans offline (not in real-time) in attempt to better understand optimal plans for complex situations. Later version of the planner may handle real time contexts.

We begin by introducing the case study system (Section IV-A) and outlining our approach and its implementation (Section IV-B). We then investigate three candidate planning scenarios.

A. Znn.com

Znn.com is a scenario self-adaptive architecture for a news website [35] commonly used in adaptive systems research [36], [37]. The system consists of a three-tiered architecture: databases store content which servers present to client-facing machines. Znn.com has three main quality aspects—response time, operating cost, and content resolution—and can modify its underlying configuration in response to changing environmental conditions.

For the purposes of this example, we select the following initial settings for the system:

- Operating cost is 30; response time is 20s.
- There are two server locations (S_1 and S_2), both initialized with two servers. Each must host a minimum of one server and a maximum of four.
- There are two databases, DB_A and DB_B both of which start in single-threaded mode and can increase to a maximum of five threads. Thread count cannot be decremented.
- Data is initially transmitted at high fidelity/resolution; we assume for simplicity that data fidelity is consistent across all servers.

The basic adaptation actions available for this system, with pre- and post-conditions, are listed in Table I. If an adaptation fails, the failure penalty is added to the plan execution time ("Time", in the table), and the rest of the system remains the same. The adaptation effects are selected to represent a hypothetical situation, but could correspond to data collected from a real deployment in future investigations.

In planning to adapt this system, then, there are four potential quality attributes of interest: system cost, system response time, content fidelity, and adaptation time (the time taken to execute the plan itself).

B. Genetic Algorithm Planning for Znn.com

We built a planner using the Java Genetic Algorithm Package (JGAP) [38], a framework that provides genetic programming capabilities. Candidate solutions are represented as trees following the grammar shown in Figure 1. The

[1]We use the terms genetic programming/genetic algorithms and GP/GA interchangeably for the purposes of this discussion.

`basicActions` are listed in Table I. This grammar is a simplified form of existing strategy specification languages, such as Stitch [39]. Those basic actions may be composed via two *control actions*, listed in Figure 1, which allow for both control flow and sequencing. Figure 2 shows a simple example plan.

We use PRISM [34], a probabilistic model checker, to evaluate candidate fitness. We performed 100,000 PRISM simulations for each candidate plan, and computed the fitness of each run according to an experiment-specific metric (assigning varying weights to different quality attributes). The plan's fitness is the average of the 100,000 scores. Infeasible plans, such as a plan that attempts to add too many servers to a location, and plans exceeding a fixed branch depth of 20, receive a fitness score of 0. The branch depth limit serves to control code bloat, a known concern in genetic algorithms [40]. Generating infeasible plans does not cause significant overhead for the system, which is dominated by fitness function execution (infeasibility checking, by contrast, is a quick operation).

A genetic algorithm iterates over populations of individual candidate solutions, which are randomly modified and recombined using mutation and crossover operators. The individuals in the initial population may be generated randomly or seeded from a starting plan; we take the latter approach in our investigations. In this prototype, only basic actions may be mutated, and they may be replaced by any other basic action from Table I. Crossover creates new candidate plans by combining partial solutions from two other candidate plans, and may be applied at any point along two trees (including at control actions). We defer to the underlying JGAP engine to set reasonable parameters for values such as mutation and crossover rates, selection tournament size, etc; we leave investigation of tuning such parameters for future work.

C. Improving an inadequate plan.

Using the approach just outlined, we first investigate whether the stochastic search-based planner can, in fact, improve a low-quality plan by optimizing for a multi-parameter fitness metric. We provided a fitness metric that strongly weights response time, with equal weight given to cost, content quality, and plan execution time. Figure 3 shows a low-quality starting plan, with a correspondingly low fitness according to our metric.

Figure 4 shows a simplified version of one result from the planner (note that population-based algorithms may generate many acceptable candidate solutions). This plan attempts to reduce text resolution, add a server at each location, and then increase the database B threads. This sequence of adaptations is very sensible in light of the provided metric, which strongly prefers a minimal response time. Indeed, the initial sequence achieves the maximum reduction in response time (from 20 to 1). The interesting complexity emerges from the use of non-deterministically successful actions and a probabilistic model checker, which led the search to include multiple failure handling branches: if the initial sequence fails at virtually any

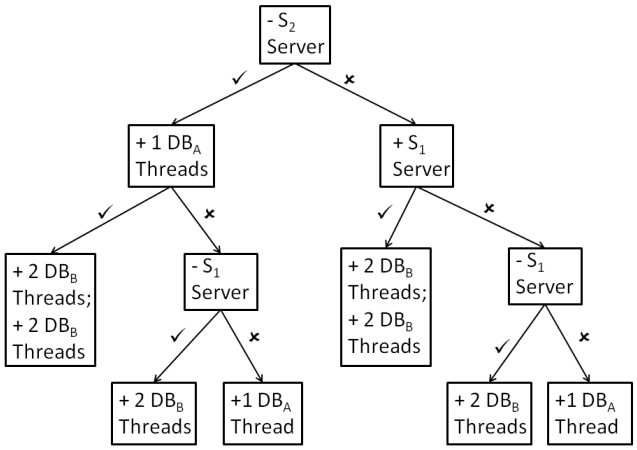

Fig. 3. Starting Plan: The plan removes a server from location 2. If successful, the plan increases the thread count for database A, otherwise it adds a server at location 1. If either of these two actions succeed, the plan tries to increase the threads at database B twice. If either of these two actions fail, the plan tries to remove a server from location 1. If the server is successfully removed from location 1, the system tries to increase the database B threads; otherwise, the plan tries to increase threads in database A.

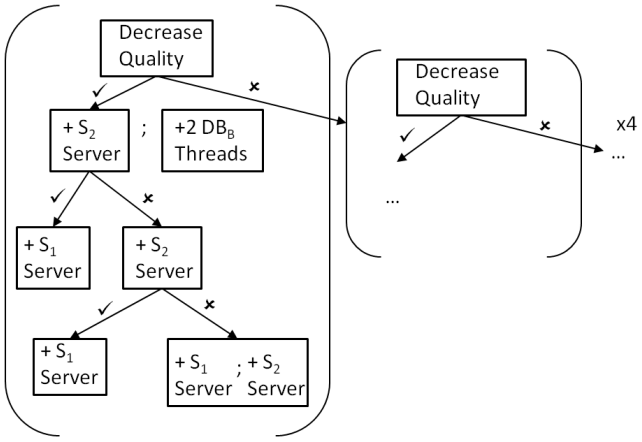

Fig. 4. Reduce Response Time Plan: The genetic program found that the best sequence of adaptations was to decrease article quality, add a server at location 1, add a server at location 2, and then add 2 database B threads . The planner also prioritized which adaptations in the sequence should be repeated if an action fails, since the plan size was limited.

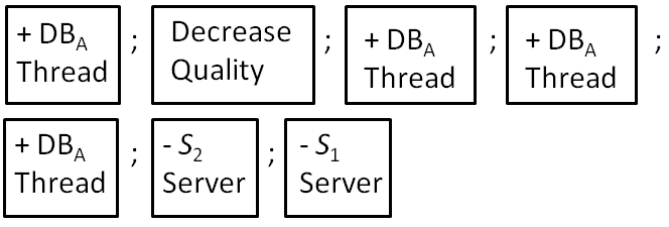

Fig. 5. Reduce Cost and Response Time Plan: This was the best plan found by the planner to reduce cost and response time. The generated plan did not adjust for possible failures, even though failures were possible.

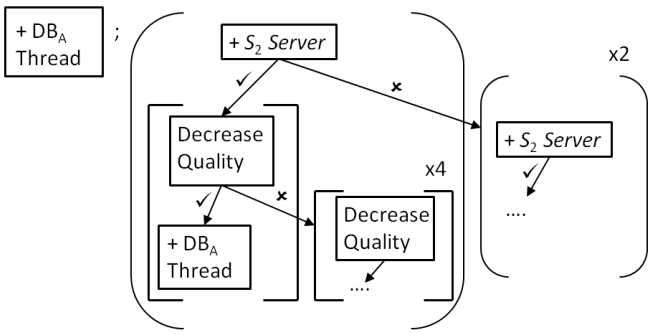

Fig. 6. Plan Generated From a Plan With a Different Metric: Starting with a plan that reduced cost and response time, the plan was altered to only reduced response time. The final plan which reduced response time is shown above. The planner found the best sequence of adaptations was to add a thread to database A, add a server at location 2, decrease article quality, and then add a thread to database A. The planner also prioritized which adaptations were to be repeated if an adaptation failed in the limited plan size.

point, the plan attempts to restart the action (with some minor order variation). Its efforts to do so are limited to 5 repetitions by the branch count limitation in the genetic algorithm.

D. Generating plans for multi-objective metrics

The metric used in the first experiment is not truly multi-objective: It strongly prefers one of the four quality dimensions over the others. In our second investigation, we provided a metric that weights cost and response time equally, preferring them strongly over the other two factors. The starting seed plan remains the same.

Figure 5 shows a plan that resulted using this metric. Note that unlike the plan generated when optimizing for response time alone, this plan is sequential, and these results were consistent across a variety of our experimental runs. This plan does reduce response time through its first 5 actions, before reducing cost by removing servers. However, unlike the previous plan, it does not add any servers, which greatly increases cost. This sequence ordering respects the linear effect that removing servers (which decreases cost but increases response time) has on the response time savings produced by the quality degradation: if the servers were removed first, the savings would decrease.

Additionally, this plan does not attempt to account for failed actions through the use of multiple conditional actions. This possibly constitutes interesting emergent behavior: multiple candidate plans strongly favoring response time alone included branching, while plans that equally emphasized two attributes consistently create sequential plans.

E. Planning for evolving quality priorities

Priorities may change over a system's life-cycle, and thus plans may need to change as well. In our final investigation, we started with the plan in Figure 5 (recall that this plan was generated via a metric that favored reduced cost and response time equally) and provided a metric that favored response time a bit more strongly than the other attributes. Figure 6 shows a simplification of the result. This plan is similar to what the planner produces for the same metric from the low quality starting plan (results not shown), and is similar (but not identical) to the plan produced for the metric that very strongly favors response time (above). Like that plan, this plan consists of a sequence of actions that maximally reduce response, with repeated reattempts in case of failure. Unlike the plans generated from the inadequate starting plan, this one starts with an increase in DB_A threads. Because of the way crossover and mutation is applied, this action is almost certainly retained unmodified from the initial plan that optimized for cost. This action has a positive impact on response time and no impact on cost, and thus the search is not incentivized to replace it. This example is a clear demonstration that the individual actions are composable with respect to the objective function, substantiating our claim that the shape of the problem is a good match with the assumptions behind this (and many) stochastic search techniques.

V. Research implications

Our initial investigations serve as a proof-of-concept of the promise of combining of advances in probabilistic model checking and stochastic search-based software engineering techniques to fully explore the quality and configuration trade-offs in one aspect of self-adaptive system design and implementation. This proof of concept raises interesting questions, with suggestions for both immediate future work and a longer-term research agenda combining these two fields.

Plan modeling. These results have possible implications for both plan behavior modeling and expressiveness. For example, when optimizing for system response time, our search-based planner aggressively accounted for failures via repeated branching sequences of adaptation actions. This is consistent with the way failures were modeled in the system: as independent actions which always had the same chance of succeeding. By performing the action multiple times, the action was likely to eventually succeed. However, action failure is not always independent. If adding a server to location 1 fails the first time, the next attempt to add a server at that location may be more likely to fail, or may always fail until the issue is resolved. Thus, more realistic results may be obtained by increasing the failure rate if the action previously failed, and accounting for these modifications in the model itself.

The planner could also be adjusted to provide alternate recovery options when an action fails and then allow the execution engine to decide if redoing the action is likely to fail. These results also support the value of including looping constructs in plan generation (such as is available in PDDL), and accounting for such loops in the model.

Unexpected interactions. One drawback to stochastic search-based techniques is that it can be difficult to explain the progression to a final result. On the other hand, their results can sometimes be interestingly unexpected. For example, as expected in a multi-objective search, our prototype generated different plans to achieve different goals. The differences in the results produced by the cost- and response time-focused

metrics are interesting because they demonstrate by example certain interactions between the two metrics as they relate to system planning and modification. While the differences between the plans was expected because of their different goals, the difference in the number of branches was unexpected. By studying such results, we can gain insight into previously unexamined interactions and features of the systems and search spaces in question. Stochastic search-based techniques can be used to quickly generate a wide diversity of different plans, configurations, or systems architectures with a variety of different quality attributes, and by examining large numbers of examples optimized to different outcomes, interesting features of property interactions (for example) may emerge that are difficult for humans to derive from first principles, even for small systems.

Stochastic search implications. Different stochastic techniques will likely be useful in different situations. Other research areas (such as computational biology) have explored the specific use cases that are appropriate for particular stochastic techniques in their domain. Truly multi-objective search techniques (such as NGSA-II or island-based GP) may be particularly beneficial in a search problem with multiple quality objectives. More work may be necessary to understand the implications (and the best ways to deploy) probabilistic fitness functions.

To understand how stochastic techniques can be applied, it would also be helpful to know how the techniques perform under different conditions. Past results have implied that certain techniques may be better for planning compared to optimizing [15]. However, a recent study successfully used genetic algorithms to adapting mobile application configuration [3], in a reasonable amount of time. The study mentions that the low ratio of feasible plans to possible adaptation combinations may greatly influence algorithm speed. Future studies could investigate how much this ratio, or other factors such as the clustering of feasible plans, affects the application of stochastic techniques to self-adaptive systems.

Leveraging diverse populations. Stochastic search techniques have shown promise for creating artificial diversity, potentially reviving the research promise of N-variant systems [41]. Diverse populations optimized for the same quality attributes (but with different configurations!) may suggest mechanisms for creating more robust systems by varying the attack or failure surface. Many population-based stochastic techniques, such as genetic programming, can be used to increase the fitness of populations of plans over time, suggesting techniques for easing reconfiguration of large sets of systems in response to new adaptations or large-scale modifications.

Interactions between MAPE components. There is still much to learn about how the different sections of the MAPE loop interact. One possibility is plans which take into account feedback from the monitor (similar to what is done in Stitch [39]). The generated plan may try an adaptation for a specific period of time and if the adaptation is not working as expected, the system will revert back to the original state to try something else. Only further investigation into these areas will allow us to fully take advantage of these beneficial interactions.

Human trust. One current problem with self-adaptive system plans is that system maintainers do not always trust generated solutions [42]. If a system begins to perform an unexpected set of actions, which may be more likely with stochastic planning techniques, a system's maintainer could easily misdiagnose a better plan as the system misbehaving. Future research needs to investigate how stochastic planners can convince human maintainers of the correct course of action when necessary. If guarantees can be provided by a planner, those guarantees can also be used to confirm the system is not misbehaving. Future research should investigate how to effectively communicate the rational behind automatically-generated self-adaptive system plans.

VI. Conclusion

The promise of software self-adaptation is the ability to deploy complex and versatile systems that can dynamically respond to changing conditions and goals. This can only be achieved by understanding how to accurately respond to changing conditions. Stochastic techniques, previously lightly explored when applied to self-adaptive systems, provide a way to greatly increase our knowledge in this area, while also providing greater flexibility in how to design and implement system changes. Both fields have advanced sufficiently to make this a particularly promising time to explore their intersection: SBSE techniques can be applied to large systems with similar problem domains, and modeling techniques can help predict and quantify self-* system behavior. We believe that multiple benefits can be gained by investigating how stochastic techniques apply to self-adaptive systems, including how quality attributes and adaptations interact and how systems can adapt to more diverse situations. As we further our knowledge of how to apply stochastic techniques to self-adaptive systems, we will help make wide-spread use of self-adaptive systems a reality.

Acknowledgement

This work is supported in part by awards N000141310401 and N000141310171 from the Office of Naval Research, the National Security Agency, and the U.S. Department of Defense through the Office of the Assistant Secretary of Defense for Research and Engineering (ASD(R&E)) under Contract HQ0034-13-D-0004. The Systems Engineering Research Center (SERC) is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology. Any views, opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Office of Naval Research, the National Security Agency, United States Department of Defense, or ASD(R&E).

References

[1] M. Harman, "The current state and future of search based software engineering," in *International Conference on Software Engineering*, 2007, pp. 342–357.

[2] B. H. C. Cheng, A. J. Ramirez, and P. K. McKinley, "Harnessing evolutionary computation to enable dynamically adaptive systems to manage uncertainty," in *Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE)*, May 2013.

[3] G. G. Pascual, M. Pinto, and L. Fuentes, "Run-time adaptation of mobile applications using genetic algorithms," in *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '13, 2013, pp. 73–82.

[4] P. Zoghi, M. Shtern, and M. Litoiu, "Designing search based adaptive systems: A quantitative approach," in *International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS 2014, 2014, pp. 7–16. [Online]. Available: http://doi.acm.org/10.1145/2593929.2593935

[5] M. Harman, Y. Jia, W. B. Langdon, J. Petke, I. H. Moghadam, S. Yoo, and F. Wu, "Genetic improvement for adaptive software engineering (keynote)," in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS 2014, 2014, pp. 1–4.

[6] *GECCO '14: Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*. ACM, 2014.

[7] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.

[8] Y. Brun, G. Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw, "Software engineering for self-adaptive systems," B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Springer-Verlag, 2009, ch. Engineering Self-Adaptive Systems Through Feedback Loops, pp. 48–70. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02161-9_3

[9] M. Salehie and L. Tahvildari, "Autonomic computing: Emerging trends and open problems," in *Proceedings of the 2005 Workshop on Design and Evolution of Autonomic Application Software*, ser. DEAS '05, 2005, pp. 1–7.

[10] ——, "Self-adaptive software: Landscape and research challenges," *ACM Trans. Auton. Adapt. Syst.*, vol. 4, no. 2, pp. 14:1–14:42, May 2009.

[11] E. Gjorven, F. Eliassen, and J. Aagedal, "Quality of adaptation," in *International Conference on Autonomic and Autonomous Systems (ICAS)*, July 2006, pp. 9–9.

[12] L. Northrop, P. Feiler, R. P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan, and K. Wallnau, "Ultra-Large-Scale Systems - The Software Challenge of the Future," Software Engineering Institute, Carnegie Mellon, Tech. Rep., June 2006.

[13] F. D. Macías-Escrivá, R. Haber, R. del Toro, and V. H. Díaz, "Self-adaptive systems: A survey of current approaches, research challenges and applications," *Expert Systems with Applications*, vol. 40, no. 7267-7279, 2013.

[14] N. Arshad, D. Heimbigner, and A. Wolf, "Deployment and dynamic reconfiguration planning for distributed software systems," in *Tools with Artificial Intelligence, 2003. Proceedings. 15th IEEE International Conference on*, Nov 2003, pp. 39–46.

[15] M. Salehie and L. Tahvildari, "Towards a goal-driven approach to action selection in self-adaptive software," *Software: Practice and Experience*, vol. 42, no. 2, pp. 211–233, 2012.

[16] A. Lapouchnian, S. Liaskos, J. Mylopoulos, and Y. Yu, "Towards requirements-driven autonomic systems design," in *Workshop on Design and Evolution of Autonomic Application Software (DEAS)*, 2005, pp. 1–7.

[17] M. Morandini, L. Penserini, and A. Perini, "Towards goal-oriented development of self-adaptive systems," in *International Workshop on Software Engineering for Adaptive and Self-managing Systems (SEAMS)*, 2008, pp. 9–16.

[18] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng, "Composing adaptive software," *Computer*, vol. 37, no. 7, pp. 56–64, Jul. 2004.

[19] B. Srivastava and S. Kambhampati, "The case for automated planning in autonomic computing," *Autonomic Computing, International Conference on*, vol. 0, pp. 331–332, 2005.

[20] R. Laddaga, "Self adaptive software problems and projects," in *IEEE Workshop on Software Evolvability*, 2006, pp. 3–10.

[21] A. Carzaniga, A. Gorla, and M. Pezzè, "Self-healing by means of automatic workarounds," in *International Workshop on Software Engineering for Adaptive and Self-managing Systems (SEAMS)*, 2008, pp. 17–24.

[22] A. Carzaniga, A. Gorla, N. Perino, and M. Pezzè, "Automatic workarounds for web applications," in *International Symposium on Foundations of Software Engineering (FSE)*, 2010, pp. 237–246.

[23] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, and R. S. Roos, "Time-aware test suite prioritization," in *Proceedings of the International Symposium on Software Testing and Analysis*, Portland, ME, USA, 2006, pp. 1–12.

[24] C. C. Michael, G. McGraw, and M. A. Schatz, "Generating software test data by evolution," *IEEE Transactions on Software Engineering*, vol. 27, no. 12, pp. 1085–1110, December 2001.

[25] S. Wappler and J. Wegener, "Evolutionary unit testing of object-oriented software using strongly-typed genetic programming," in *Genetic and Evolutionary Computation Conference*, 2006, pp. 1925–1932.

[26] A. Barreto, M. Barros, and C. Werner, "Staffing a software project: a constraint satisfaction approach," *Computers and Operations Research*, vol. 35, no. 10, pp. 3073–3089, 2008.

[27] C. Le Goues, T. Nguyen, S. Forrest, and W. Weimer, "GenProg: A generic method for automatic software repair," *IEEE Transactions on Software Engineering*, vol. 38, pp. 54–72, 2012.

[28] D. Kim, J. Nam, J. Song, and S. Kim, "Automatic patch generation learned from human-written patches," in *Proceedings of the 35th International Conference on Software Engineering*, San Francisco, CA, USA, 2013, pp. 802–811.

[29] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 11:1–11:61, Dec. 2012.

[30] J. R. Koza, "Genetic programming: On the programming of computers by means of natural selection, 1992," *See http://miriad. Iip6. fr/microbes Modeling Adaptive Multi-Agent Systems Inspired by Developmental Biology*, vol. 229, 1992.

[31] C. Le Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer, "A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each," in *International Conference on Software Engineering*, 2012, pp. 3–13.

[32] M. Harman, W. B. Langdon, and W. Weimer, "Genetic programming for reverse engineering," in *Working Conference on Reverse Engineering*. IEEE, 2013, pp. 1–10, invited Keynote.

[33] J. Petke, M. Harman, W. B. Langdon, and W. Weimer, "Using genetic improvement and code transplants to specialise a C++ program to a problem class," in *European Conference on Genetic Programming*. Springer, 2014, pp. 137–149.

[34] M. Kwiatkowska, G. Norman, and D. Parker, "Prism 4.0: Verification of probabilistic real-time systems," in *Proceedings of the 23rd International Conference on Computer Aided Verification*, ser. CAV'11, 2011, pp. 585–591.

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

[36] J. Cámara, G. A. Moreno, and D. Garlan, "Stochastic game analysis and latency awareness for proactive self-adaptation," in *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2014, pp. 155–164.

[37] E. Yuan, S. Malek, B. Schmerl, D. Garlan, and J. Gennari, "Architecture-based self-protecting software systems," in *Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures*, ser. QoSA '13, 2013, pp. 33–42.

[38] "JGAP:java genetic algorithms package," http://www.jgap.souceforge.net, accessed: 2015-01-21.

[39] S.-W. Cheng and D. Garlan, "Stitch: A language for architecture-based self-adaptation," *J. Syst. Softw.*, vol. 85, no. 12, pp. 2860–2875, Dec. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2012.02.060

[40] S. Gustafson, A. Ekart, E. Burke, and G. Kendall, "Problem difficulty and code growth in genetic programming," *Genetic Programming and Evolvable Machines*, pp. 271–290, 2004.

[41] B. Cox, D. Evans, A. Filipi, J. Rowanhill, W. Hu, J. Davidson, J. Knight, A. Nguyen-Tuong, and J. Hiser, "N-variant systems: a secretless framework for security through diversity," in *USENIX Security Symposium*, 2006.

[42] J. A. McCann, R. De Lemos, M. Huebscher, O. F. Rana, and A. Wombacher, "Can self-managed systems be trusted?: Some views and trends," *Knowl. Eng. Rev.*, vol. 21, no. 3, pp. 239–248, Sep. 2006.