Planning as Optimization: Dynamically Discovering Optimal Configurations for Runtime Situations

*Authors ordered alphabetically

Erik M. Fredericks*, Ilias Gerostathopoulos†, Christian Krupitzer‡, and Thomas Vogel§

**Oakland University*, Rochester, MI, USA. Email: fredericks@oakland.edu

†*Technical University of Munich*, Munich, Germany. Email: gerostat@in.tum.de

‡*University of Würzburg*, Würzburg, Germany. Email: christian.krupitzer@uni-wuerzburg.de

§*Humboldt-Universität zu Berlin*, Berlin, Germany. Email: thomas.vogel@cs.hu-berlin.de

***Abstract*—The large number of possible configurations of modern software-based systems, combined with the large number of possible environmental situations of such systems, prohibits enumerating all adaptation options at design time and necessitates planning at run time to dynamically identify an appropriate configuration for a situation. While numerous planning techniques exist, they typically assume a detailed state-based model of the system and that the situations that warrant adaptations are known. Both of these assumptions can be violated in complex, real-world systems. As a result, adaptation planning must rely on simple models that capture what can be changed (input parameters) and observed in the system and environment (output and context parameters). We therefore propose planning as optimization: the use of optimization strategies to discover optimal system configurations at runtime for each distinct situation that is also dynamically identified at runtime. We apply our approach to CrowdNav, an open-source traffic routing system with the characteristics of a real-world system. We identify situations via clustering and conduct an empirical study that compares Bayesian optimization and two types of evolutionary optimization (NSGA-II and novelty search) in CrowdNav.**

***Index Terms*—planning, optimization, Bayesian optimization, evolutionary search, traffic routing model problem**

I. Introduction

A self-adaptive system (SAS) continuously monitors itself and its environment to ensure that, for each environmental situation, a valid system configuration is applied that achieves optimal performance and behavior [1]. Using the monitored data, adaptation planning aims at identifying such a valid and optimal configuration. For this purpose, a SAS integrates decision metrics based on rules or models with higher degrees of freedom [2]. However, both categories have their shortcomings. Rules can be inflexible if not accompanied with runtime learning [3] and they cannot cover all situations due to the state-space explosion related to the number of possible situations and configurations [1], [4], [5]. Models such as Discrete Time Markov Chains [6] might cope with higher numbers of situations and configurations as they offer more adaptation freedom. However, such models require detailed knowledge about the internal behavior of the system, the behavior of its environment, and the effects of adaptation actions to the system for all possible environmental situations.

In complex, real-world systems it is difficult to both identify the environmental situations that warrant adaptation and understand how changes to the system affect the performance and behavior in these situations. For example, optimizing a router in a traffic system requires a detailed model of the behavior of the individual cars, the traffic events that may occur (e.g., increase of traffic demand), and how changes in routing affect the performance (e.g., the average trip time) and behavior (e.g., traffic jams). It is a challenge to obtain, maintain, and tailor such detailed models to each environmental situation.

An alternative approach is to model the system as a *black-box* with input and output parameters and its environment as a set of context parameters. Optimizing the system then involves finding values for input parameters (i.e., *configurations*) that optimize the system performance and behavior specified in terms of output parameters, and do so for each environmental situation specified in terms of context parameters. In other words, optimization is performed as a means of adaptation planning for each situation. To be effective as a method at runtime and applicable to real-world systems, optimization needs to cope with large numbers and ranges of input, output, and context parameters, and provide useful results in a timely manner. The last point is especially important for usage in scenarios where the optimization horizon is short.

In this setting, different optimization techniques can be used if they can cope with problems that are (i) *black-box*, i.e., the function relating the input to the output parameters is unknown, (ii) *high-dimensional*, i.e., large number and ranges of input, output, and context parameters, and (iii) *expensive*, i.e., they need to be solved in a minimal number of iterations since computing outputs may be costly in terms of time or other resources [7]. Examples of such optimization techniques include Bayesian and evolutionary approaches.

Another challenge is "to support self-adaptation for complex types of uncertainties" [8, p. 436], i.e., when it is not possible to model *a priori* the situations in which a SAS might reside at run time. In theory, optimization at run time can solely focus on finding a system configuration that works well in any situation (e.g., to optimize the parameters of a web server without considering the fluctuations in demand). In practice, such a situation-agnostic optimization may lead to sub-optimal configurations. Therefore, a system should dynamically identify the *distinct* situations it encounters at runtime, based on the effect of contextual parameters on the outputs, and optimize individually for each of them. For example, a distinct situation in a traffic system may be a traffic jam or an accident.

Responding to these challenges, we propose an approach that we call *planning as optimization*: the use of optimization strategies to discover optimal system configurations at runtime for each distinct situation that is dynamically identified at runtime. Our approach draws inspiration from online optimization and learning [9]–[11] and tackles complex systems that are modeled as black-box, high-dimensional, and computationally expensive optimization problems.

In particular, we make the following contributions:

- We show the feasibility of planning as optimization by dynamically identifying distinct situations at runtime via clustering and by discovering optimal configurations via different optimization techniques.
- We compare the solution quality, convergence, and overhead of three optimization techniques in an empirical study using CrowdNav [12], an open-source self-adaptation exemplar of a traffic system that corresponds to a black-box, high-dimensional, and expensive optimization problem.

As optimization techniques we select Bayesian optimization and two evolutionary optimization techniques (NSGA-II and novelty search) as they can cope with such a problem. Since in general no technique is superior to any other technique for any given optimization problem (*cf.* no "free lunch" theorems for optimization [13]), we performed the comparison between the three techniques to identify which performs best in CrowdNav. This is also a first step towards our vision of dynamically identifying and using the best optimizer at runtime.

The rest of this paper is organized as follows: In Section II, we illustrate the research gap with a literature review. Section III describes our motivating scenario based on CrowdNav. We present our planning as optimization approach involving situation detection and optimization with three optimizers in Section IV. We evaluate our approach in an empirical study in Section V and discuss remaining research challenges in Section VI. Lastly, we summarize our findings in Section VII.

II. Using Optimization in Self-adaptive Systems

Many approaches apply optimization in SASs for adaptation planning by generating new system configurations or adaptation plans. We analyzed these techniques based on approaches published during the last ten years in conferences and journals related to SASs.[1] As listed in Table I, we identified the use of 29 different techniques in 51 publications. This list shows that a large set of techniques from different classes such as probabilistic, combinatorial, evolutionary, stochastic, mathematical, and meta-heuristic optimization are applied in SASs. However, there is less information on how the techniques compare to each other in terms of assumptions, overhead, and quality of the achieved solutions.

Our overall approach of finding optimal configurations in different situations bears similarities with the work by Porter *et al.* on learning optimal system configurations in emergent software systems [9]. This work focuses on providing a general framework for online learning that interleaves exhaustive search of configurations with determination of environmental situations. In contrast, our work focuses on optimization problems of higher dimensionality that require more sophisticated search techniques. Similar to our work, Kinneer *et al.* [14] introduced an approach for adaptation planning that re-uses the knowledge of existing plans for optimization via genetic algorithms. In contrast to our approach, the authors rely on tactics that can change the system by integrating explicit knowledge of the system, whereas we target black-box systems where such knowledge is not available. Further, different authors did studies to compare the performance of optimization techniques. Bischl *et al.* compare mlrMBO, a flexible toolbox for black-box optimization with Bayesian optimization [15] against other optimization techniques, including NSGA-II. However, they used theoretical problems for their comparison whereas we apply the optimization techniques in a (simulated) traffic system that represents a real-world problem. In another study, Moreno *et al.* [16] compare Markov Decision and Analytic Hierarchy Processes for adaptation planning in *RUBiS*. However, both techniques are not applicable to black-box systems that we are targeting.

TABLE I
List of Optimization techniques used in SASs.

Probabilistic Optimization Techniques Bayesian Networks, Bayesian Optimization, Simulated Annealing
Combinatorial Optimization Techniques Cross-entropy Method for Combinatorial Optimization, Decentralized Combinatorial Optimization
Evolutionary Optimization Techniques Evolutionary Algorithm, Genetic Algorithm, Genetic Programming, Learning Classifier System, NSGA-II, SPEA2
Stochastic Optimization Techniques Greedy Algorithm, Markov Decision Process, Stochastic Approximation, Stochastic Programming, Variable Neighbourhood Search
Mathematical Optimization Techniques Binary Programming, Integer Programming, Linear Programming, Sequential Quadratic Programming, Convex Optimization Solver, Pattern Search Algorithm
Meta-Heuristic Optimization Techniques Heuristic Algorithm, Tabu Search
Other Optimization Techniques Canonical Correlation, Weighted Sum Model, Reinforcement Learning, Distributed Constraint Optimization, Gradient Descent

III. Motivating Scenario

Many systems have been modeled as SASs in domains such as cyber-physical [17], [18] and intelligent traffic systems [19], [20]. In this paper, we focus on intelligent traffic systems and perform an empirical study on the CrowdNav[2] exemplar [12], a SAS that performs smart routing for city-wide traffic management. CrowdNav combines the SUMO traffic simulator[3] with a custom-built module for routing. In CrowdNav, cars continuously drive in the city of Eichstädt from a randomly selected destination to another with routes provided by the routing

[1]We considered ACM TAAS, ICAC, SASO, SEAMS, and FSE. Due to space constraints we did not include the references in this paper but published them here: https://doi.org/10.5281/zenodo.2584266.

[2]CrowdNav: https://github.com/iliasger/CrowdNav

[3]Simulation of Urban MObility: http://sumo.dlr.de/index.html

TABLE II
INPUT, OUTPUT, AND CONTEXT PARAMETERS OF CROWDNAV.

Input Parameters			
Name	**Type**	**Range**	
route randomization	float	[0-0.3]	Controls the random noise introduced to avoid giving the same routes
exploration percentage	float	[0-0.3]	Controls the ratio of smart cars used as explorers
static info weight	float	[1-2.5]	Controls the importance of static information (i.e., max speed, street length) on routing
dynamic info weight	float	[1-2.5]	Controls the importance of dynamic information (i.e., observed traffic) on routing
exploration weight	integer	[5-20]	Controls the degree of exploration of the explorers
data freshness threshold	integer	[100-700]	Threshold for considering traffic-related data as stale and disregard them
re-routing frequency	integer	[10-70]	Controls how often the router should be invoked to re-route a car
Output Parameters			
Name	**Type**	**Constraints**	**Description**
trip overhead	float	>1	The actual trip time divided by the theoretical trip time if a car travels at max speed
routing cost	integer	>0	The time needed by the router to re-route all cars
Context Parameter			
Name	**Type**	**Constraints**	**Description**
number of cars	integer	>0	The total number of cars that are in the city and use the smart configurable router

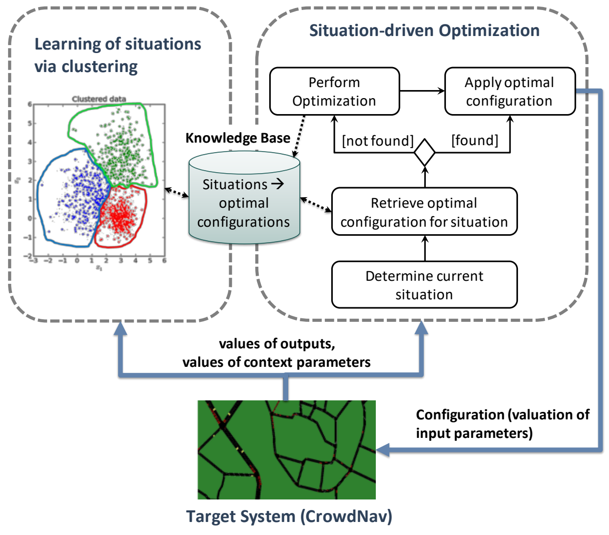

Learning of situations via clustering
Clustered data
Knowledge Base
Situations → optimal configurations
Situation-driven Optimization
Perform Optimization
Apply optimal configuration
[not found]
[found]
Retrieve optimal configuration for situation
Determine current situation
values of outputs, values of context parameters
Configuration (valuation of input parameters)
Target System (CrowdNav)

Fig. 1. Overview of our approach.

module. The module is used by all cars and comprises seven configurable numeric input parameters detailed in Table II. Moreover, CrowdNav provides two outputs, *trip overhead* and *routing cost*, and features one context parameter that can be observed but not controlled: the *number of cars* (see Table II).

Selecting an optimal configuration in CrowdNav entails providing a value for each input parameter (*route randomization*, *exploration percentage*, etc.) with the goal of minimizing *trip overhead* and *routing cost* for each situation determined by the *number of cars*. Note that a configuration impacts each output parameter differently, leading to a multi-objective optimization problem with competing concerns.

Viewed as a numeric optimization problem, CrowdNav optimization has some distinct properties. First, it is a black-box optimization problem since there is no known model/function that relates inputs to outputs. As a result, to evaluate a configuration, it needs to be applied to CrowdNav and its effects need to be measured on the outputs. Second, the number and range of input and context parameters in CrowdNav also creates a large configuration space, resulting in a high-dimensional problem. Given that a float is represented in Python by default with 15 decimal points and assuming that no discretization is performed, the input configuration space of CrowdNav is $0.3 \cdot 10^{15} \cdot 0.3 \cdot 10^{15} \cdot 1.5 \cdot 10^{15} \cdot 1.5 \cdot 10^{15} \cdot 15 \cdot 600 \cdot 60 \approx 100,000 \cdot 10^{60}$. Third, the outputs of CrowdNav exhibit high variance (*noisy* outputs). To compare different configurations as to their effects on the outputs, multiple samples of the outputs are required to ensure that statistical measures are robust to noise and outliers. We found that 5000 samples are sufficient to characterize a situation and to evaluate a configuration. However, collecting multiple samples from a running system increases the time needed to evaluate a configuration, which makes CrowdNav an expensive optimization problem.

Finally, optimization of CrowdNav must consider different situations that depend on the value of its context parameter, *number of cars*, as it is unlikely that a single optimal configuration exists for different values (e.g., 300 vs. 800 cars).

IV. APPROACH

Our approach aims at optimizing systems that are modeled as black-box, high-dimensional, and computationally expensive optimization problems in different runtime situations. Therefore, there is a need for both learning the distinct situations the system might be in, and optimizing the system for each such situation. Our approach comprises two modes, each dealing with one of the above-mentioned needs (Figure 1).
Mode #1: Learning of situations via clustering. In this mode, the outputs of the target system and the context parameters are monitored. The goal is to determine the valuations of context parameters that can be grouped together in a distinct *situation* in terms of the outputs. Once these valuations are learned, the system can detect its current situation by only monitoring its context parameters.

To learn a situation, we assume that for each context parameter, a discrete number of ranges for its values is provided. For instance, the designers of CrowdNav may specify that the *number of cars* – a context parameter in CrowdNav – belongs to one of the ranges [0,100], [100, 200], [200, 300], [300, ∞). All possible states that the system can reside in is the Cartesian product of the ranges of all of its context parameters. While operating, the system can traverse from one state to another. By observing the values of its output parameters for each state, our approach essentially groups together states into situations by learning which states are similar to each other with respect to their impact on the output parameters. The clustering of the output data to learn situations is detailed in Section IV-A.

Mode #2: Situation-driven optimization. In the second mode, the context parameters of the target system are monitored and the clustering model learned in *Mode #1* is used to determine the current situation. If the current situation is different from the previous one, the `Knowledge Base` (*cf.* Figure 1) is queried for an optimal configuration. If such a configuration is already known, it is applied to the target system. If not, an optimization process starts with the goal of identifying an optimal configuration for the current situation, applying the optimized configuration to the target system, and saving it to the `Knowledge Base` for future use.

Situation-driven optimization relies on the learning of situations via clustering to optimize for distinct situations. If clustering is not performed or returns just a single situation (indicating that the context does not influence the outputs), optimization can still be performed for this general case.

We assume that an optimization process is not interrupted once started so that it finishes before the current situation changes. If this assumption does not hold, the system needs to be equipped with a mechanism of saving the currently best configuration for a situation to the `Knowledge Base` and continuing the optimization process when it is next detected. Such an incremental optimization is a topic of future work.

The optimization process of our approach can be guided by different optimization techniques. So far, we considered three state-of-the-art techniques in optimizing CrowdNav (*cf.* Section IV-B). Generally, the choice of the technique highly depends on the target system and in particular in the response surface of its outputs. We therefore provide a basis for comparison between multiple optimization techniques (*cf.* Section V).

We next discuss how distinct situations are learned via clustering as well as the optimization techniques we use.

A. Clustering-based Situation Learning

To group individual context states to situations, our approach continuously observes both the valuation of context parameters and the corresponding system outputs. For each context state (the number of context states is the Cartesian product of all the possible ranges of context parameters), a number of observations of system outputs are collected. This collection results in a dataset for each context state, with features computed for each dataset. Possible features include well-known statistical measures of central tendency and dispersion such as arithmetic mean, median, variance, and standard deviation. Our approach assumes that such features are provided for each system output (however, we can always use generic features, such as the statistical measures mentioned above). The features for each context state are then fed to a clustering algorithm – we use *k-means* [21] – that determines the datasets that are most similar and should form a cluster. In particular, given a number of clusters *k*, *k-means* iteratively tries to find a centroid for each cluster so that the sum of the squared Euclidean distances between an observation assigned to the cluster and the cluster's centroid is minimized. The context states that correspond to the datasets belonging to the same cluster are then grouped together in a situation.

For illustration with CrowsNav, consider that the arithmetic mean and median of *trip overheads* are used as clustering features. Moreover, the only context variable is the *number of cars* that can be in one of the four ranges [0,100], [100, 200], [200, 300], [300, ∞) corresponding to four system states. Our clustering approach will compute the mean and median of a large number of samples (5000) of trip overhead for each state. If *k-means* groups the first three states into a single cluster, then there is not enough difference between having 100 or 300 cars, corresponding to a single situation (e.g., "low traffic").

A problem with using *k-means* or any other clustering algorithm at runtime is that these algorithms typically expect the *number of clusters* to be provided by the user. Instead, we assume that a list of candidate numbers of clusters is provided, from which the optimal number of clusters is automatically determined at runtime based on the available data. To determine the optimal number of clusters, we follow the Silhouette method [22], a well-accepted method for measuring clustering validity [23]. For each number of clusters, we perform clustering via *k-means* and then compute the average Silhouette coefficient. In particular, the Silhouette coefficient sc for a datum is calculated by Equation 1:

$$sc = \frac{b - a}{max(a, b)} \tag{1}$$

where a is the average Euclidean distance between the datum and other data in its cluster and b is the average Euclidean distance between the datum and other data in the next nearest cluster. A Silhouette coefficient takes values within [-1,1] with values close to 1 indicating a good match of the datum to the cluster. The average Silhouette coefficient is calculated by considering all data points and provides a measure of how well the data are assigned to clusters [23]. In our approach, we select the number of clusters that yields the highest average Silhouette coefficient.

B. Optimization Techniques

Since each optimization technique presented in Section II has its own strengths and limitations, they cannot all be applied to all optimization problems. For a black-box problem/system, there is generally no model describing its function/behavior. This limitation rules out all techniques that require such a model, such as linear programming. Moreover, a problem with high dimensionality poses challenges for techniques that do not scale and a multi-objective problem prevents the use of single-objective techniques.

For this paper, we consider probabilistic and evolutionary techniques to optimize CrowdNav, as it is a black-box system comprising high-dimensionality and multi-objective characteristics. These techniques have been shown to cope with such criteria in SASs (*cf.* Section II) in related optimization problems. Particularly, as a probabilistic technique we use *Bayesian Optimization* that has been previously applied to CrowdNav for a single-objective problem [24] and as evolutionary techniques we use the widespread *NSGA-II* algorithm and *novelty search*. These three techniques all rely on some form of *fitness functions* to evaluate a configuration of the CrowdNav router in terms of its objectives: *trip overhead* and *routing cost*. Having no model of CrowdNav, we have no means to directly calculate the fitness of a router configuration. Instead, a configuration is applied to CrowdNav and its effects on the *trip overhead* and *router cost* are measured to obtain the fitness. This corresponds to an *online experiment* of applying and evaluating a router configuration in the running CrowdNav. Next, we briefly describe the three optimization techniques we employed.

1) Bayesian Optimization: Bayesian or sequential model-based optimization is an approach to global optimization that can be used for efficiently optimizing expensive black-box functions [25]. By expensive it is meant that a single evaluation of the function is costly in terms of time or resources. Bayesian optimization can be used for optimizing a single objective (already demonstrated on CrowdNav [24]) as well as a multi-objective problem, which is the focus of this paper.

In short, Bayesian optimization works as follows. Given a number of execution steps (*budget*), at each step, the process fits a regression model to the selected inputs and obtained outputs, then uses the model to propose a promising set of inputs to try next by optimizing an *acquisition function*. A common approach for the regression model is to use Gaussian processes. Such processes can capture the uncertainty in the measurements and deal with noisy functions. They have been applied to CrowdNav whose outputs have high variance [24].

Being a topic of active research, many different flavors of Bayesian multi-objective optimization have been proposed [26]. They differ mainly on whether they use a single (i.e., scalarized objectives) or separate regression models. We have chosen a variant called *$\mathcal{S}$-Metric-Selection-based Efficient Global Optimization* (SMS-EGO) [27]. In this algorithm, separate regression models for each objective are fitted and the proposed point to evaluate next is selected based on the estimated contribution to the hypervolume indicator.

2) NSGA-II: The Non-dominated Sorting Genetic Algorithm II (NSGA-II) is a multi-objective evolutionary algorithm that searches for pareto-optimal solutions to an optimization problem [28]. During the search, NSGA-II evolves a population of candidate solutions using crossover, mutation, and selection operators inspired by evolution and natural selection in biology. The goal is to find solutions that are optimal with respect to the search objectives. For this purpose, a fitness function is used that evaluates how well a solution satisfies the objectives. The resulting fitness of a solution determines the selection of this solution to the next generation for further evolution steps. Having multiple objectives, the result of the search is a pareto frontier, a set of solutions with the best trade-offs between the objectives that could be found.

Additionally, NSGA-II promotes the diversity of solutions, which supports exploring the search and objective spaces. In contrast to the original NSGA, it introduces elitism that avoids losing good solutions during the search and improves the performance of the non-dominated sorting. These aspects make NSGA-II a popular technique that is widely used in optimization as well as search-based software engineering [29].

For CrowdNav, a configuration of the router (i.e., a candidate solution) is encoded as a chromosome or a vector of components with one component for each input parameter. While mutation randomly modifies the value of one input parameter taking the defined range of this parameter into account, crossover recombines two configurations to obtain a new configuration. To evaluate the fitness of a single configuration in terms of *trip overhead* and *routing cost*, an online experiment is performed in the running CrowdNav.

3) Novelty Search: Novelty search provides an alternative method for evolutionary optimization by searching the solution space for uniquely optimal, rather than solely optimal, solutions. In contrast to more common evolutionary processes (e.g., genetic algorithms), novelty search relies on a measure of distance between genomes as a point of optimization while considering the validity and/or optimality of the genomes, commonly called the *novelty metric*. The intent of the novelty function is to avoid the issue in which an evolutionary process becomes "stuck" in a local optima and instead explores the solution space for a globally-optimal solution [30].

Generally, the novelty metric is calculated from a combination of the combined pair-wise distances between all generated solutions (e.g., Manhattan distance between instantiated genome parameters) and a measure of performance for each solution (e.g., the fitness of solutions). These values are combined via a linear-weighted sum into the overall novelty metric that quantifies the diversity, combined with the optimality, of each solution. For this instantiation of novelty search, we use the same approach for defining genomes as does our implementation of NSGA-II.

Novelty search also differs from common evolutionary approaches in that it maintains a *novelty archive* of the most diverse solutions. This archive is populated each generation by ranking all genomes in the population, as well as the contents of the archive, and selecting the k most diverse solutions (i.e., the solutions with the highest novelty score). Upon completion, the novelty archive will contain the k most diverse solutions discovered throughout the entirety of the search. For this paper, we set k to retain the top 20% of all evaluated solutions.

V. Evaluation

In this section, we evaluate the two modes of our approach, learning of situations and situation-driven optimization.

A. Experimental Setup

Using CrowdNav as a managed system, we investigate the learning of situations by clustering and how different optimization techniques for adaptation planning perform in identifying optimal configurations for these situations. The

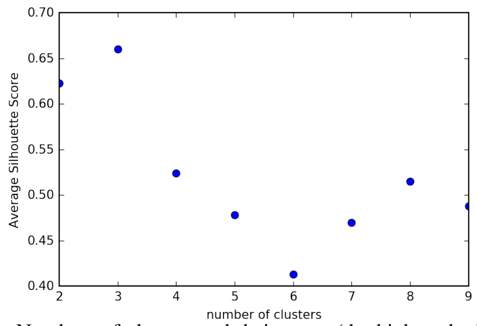


Fig. 2. Numbers of clusters and their score (the higher, the better).

optimization techniques are Bayesian Optimization, NSGA-II, and novelty search. To connect the optimization techniques to CrowdNav, we use *RTX*,[4] a framework that supports online experiments with CrowdNav. Thus, the three optimization techniques have been implemented in RTX.[5]

We ran the experiments comparing the different optimization techniques on identical virtual machines.[6] For each experiment, we use the Wilcoxon-Mann-Whitney U-test ($p < 0.05$) to determine statistical differences between datasets, as we assume no normality of data. To establish statistical significance, we performed 30 replicates for each experiment [31], where the replicate served as the seed value to RTX/CrowdNav.

B. Experiments on Learning of Situations via Clustering

To show the feasibility of dynamically identifying distinct situations at runtime via clustering, we performed an empirical study with CrowdNav, in which we experimented with different values of the context parameter *number of cars*. Assuming that the ranges provided to the method for this parameter are [100-150], (150-200], ..., (750-800], we performed 14 experiments where each experiment had 150, 200, 250, ..., 750, 800 number of cars respectively, assuming that the highest value of a range is representative of all other values in the range. For each experiment, we collected 5000 samples of *trip overhead* and calculated the average, median, 75th percentile, 90th percentile, standard deviation, and variance of each dataset. Then, we invoked *k-means* with all the features for each dataset required to produce clusters with numbers in the range [2,9]. For each of the 8 produced clusterings, we computed the average Silhouette score. The results are depicted in Figure 2. As can be seen, the clustering with three clusters had the highest score and thus was selected by our approach. Such a clustering groups the values of the context parameter into the following three groups:

- Cluster/Situation #1: Car counts between 101 and 500.
- Cluster/Situation #2: Car counts between 501 and 700.
- Cluster/Situation #3: Car counts between 701 and 800.

These three clusters correspond to "low traffic," "medium traffic," and "high traffic," respectively. We note here that an extension of these experiments to also consider the features of *routing cost* is straightforward. Next, we explain how our approach tries to optimize both CrowdNav objectives in each of the above situations by setting the number of cars to its highest possible value for each situation (i.e., 500, 700, and 800 cars, respectively).

C. Experiments on Situation-driven Optimization

To show the feasibility of discovering optimal configurations via different optimization techniques and evaluate these techniques in terms of solution quality, convergence, and overhead, we performed the following empirical study with CrowdNav. For each of the three distinct situations of 500, 700, and 800 cars (*cf.* previous section), we compare the three optimization techniques – Bayesian optimization (BOGP), NSGA-II, and novelty search – with each other and with random search as a baseline (*cf.* [31]). We configured each technique to perform 100 fitness evaluations, that is, 100 router configurations were generated, applied, and evaluated on CrowdNav during the optimization process. For BOGP, this corresponds to a budget of 100. For NSGA-II and novelty search, a population of size 10 is evolved over 10 generations with an offspring size of 10. Thus, each member of the population is adapted by mutation or crossover in each generation, resulting in a total of 100 candidate solutions evaluated throughout the search. The crossover and mutation rates are set to 0.7 and 0.3, respectively. Moreover, for novelty search the novelty archive size is set to 20%. To provide a fair comparison, random search evaluated 100 randomly-generated candidate solutions.

Given that an evaluation of a single configuration takes approximately six minutes in our setting, if ten configurations are evaluated in parallel (e.g., for NSGA-II evaluating all ten configurations of one generation concurrently), then 100 evaluations can be performed within a given optimization horizon of 60 mins. In our experiments, we did not employ any parallelization to better track the overall process. We now discuss the results from 30 replicates of running each optimization technique for each of the three situations.

1) Solution Quality: To evaluate how well an optimization technique performs, we consider the quality of the pareto-optimal router configurations found by the technique. The quality of a configuration is measured by how well the objectives, *trip overhead* and *routing cost*, are minimized.

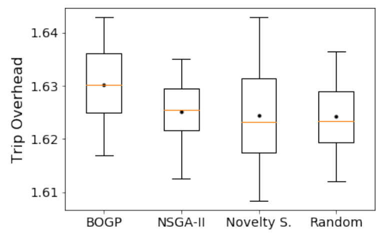
Considering each objective individually, we selected the minimum value of *trip overhead* and *routing cost* achieved by the technique's pareto-optimal configurations. The corresponding trip overheads and routing costs over the 30 replicates are plotted in Figures 3(a)-(f) for each of the three distinct situations of 500, 700, and 800 cars. The averages and medians of trip overhead and routing cost are also listed in Table III showing that the average and median values do not differ much. Thus, we just use the average values in the following.

For all situations, random search, NSGA-II, and novelty search found configurations that achieve similar average trip overheads of around 1.62 (500 cars), 1.81 (700 cars), and 1.91 (800 cars) while BOGP is slightly worse with values of 1.63, 1.82, and 1.92, respectively. Concerning the average

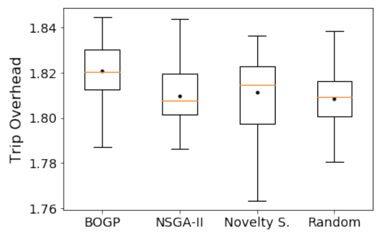
[4]RTX is available open source: https://github.com/iliasger/RTX/

[5]https://github.com/iliasger/RTX/tree/saso19

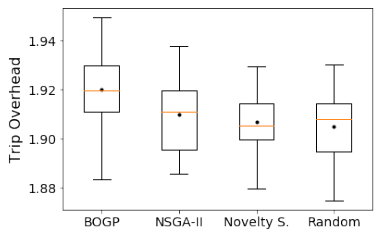
[6]The virtual machines run Ubuntu 18.04, 16 Intel Haswell vCPUs, and 14.4gb of memory, and they host CrowdNav in version https://github.com/iliasger/CrowdNav/tree/saso19 with SUMO in version 0.32.0.



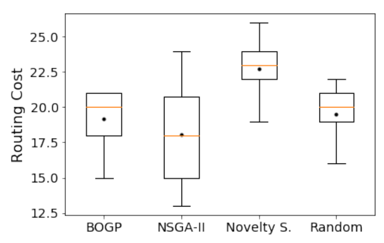
(a) Trip Overhead for 500 cars.



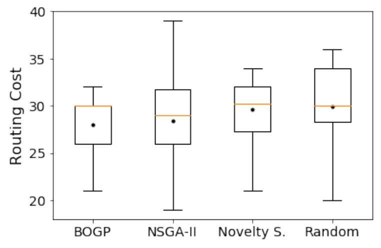
(b) Trip Overhead for 700 cars.



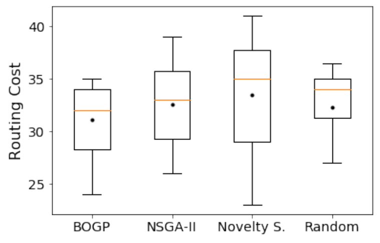
(c) Trip Overhead for 800 cars.



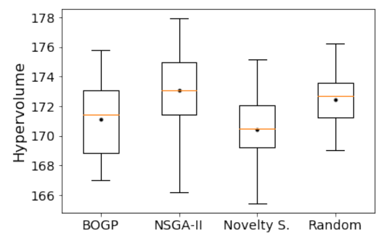
(d) Routing Cost for 500 cars.



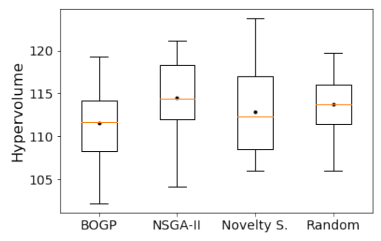
(e) Routing Cost for 700 cars.



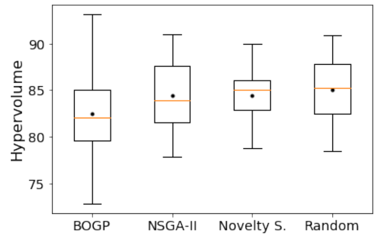
(f) Routing Cost for 800 cars.



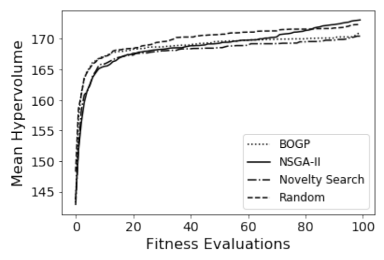
(g) Hypervolume for 500 cars.



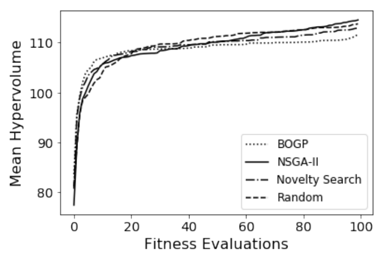
(h) Hypervolume for 700 cars.



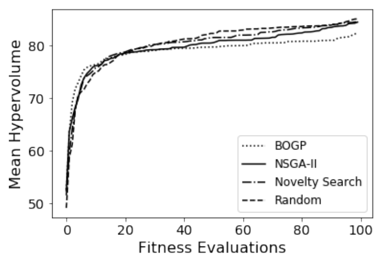
(i) Hypervolume for 800 cars.



(j) Hypervolume Evolution for 500 cars.



(k) Hypervolume Evolution for 700 cars.



(l) Hypervolume Evolution for 800 cars.

Fig. 3. Trip Overheads & Routing Costs (the lower, the better), and Hypervolumes (the higher, the better) for 500, 700, and 800 cars. Dots represent averages.

TABLE III
TRIP OVERHEADS, ROUTING COSTS, AND HYPERVOLUMES.

	Technique	Trip Overhead		Routing Cost		Hypervolume	
		Average	Median	Average	Median	Average	Median
500 cars	BOGP	1.6302	1.6301	19.20	20.00	171.1091	171.4032
	NSGA-II	1.6250	1.6254	18.03	18.00	173.0898	173.0635
	Novelty Search	1.6244	1.6232	22.73	23.00	170.4439	170.4801
	Random	1.6242	1.6234	19.53	20.00	172.4161	172.6764
700 cars	BOGP	1.8206	1.8202	28.03	30.00	111.6234	111.6794
	NSGA-II	1.8095	1.8078	28.45	29.00	114.5563	114.4319
	Novelty Search	1.8112	1.8148	29.62	30.25	112.9054	112.2987
	Random	1.8086	1.8093	29.90	30.00	113.7743	113.7692
800 cars	BOGP	1.9201	1.9195	31.07	32.00	82.4267	82.0824
	NSGA-II	1.9098	1.9108	32.53	33.00	84.3859	83.9632
	Novelty Search	1.9068	1.9052	33.50	35.00	84.4170	85.0677
	Random	1.9049	1.9079	32.32	34.00	85.0024	85.2451

routing cost in the situation with 500 cars, NSGA-II obtained the best results (18.03), followed by BOGP (19.20), Random (19.53), and novelty search (22.73). In contrast, for 700 cars, BOGP discovered the best results (28.03) followed by NSGA-II (28.45), novelty search (29.62), and random search (29.90). Likewise for 800 cars, BOGP found again the best configurations (31.07) followed by random search (32.32), NSGA-II (32.53), and novelty search (33.50).

Using the Wilcoxon-Mann-Whitney U-test ($p < 0.05$) and concerning the *trip overhead*, we observe a statistically significant difference for 500 and 700 cars between NSGA-II and BOGP, novelty search and BOGP, as well as random search and BOGP; and for 800 cars between novelty search and BOGP, as well as random search and BOGP. Concerning the *routing cost*, a statistically significant difference exists for 500 cars between novelty search and BOGP, novelty search and NSGA-II, random search and NSGA-II, as well as random search and novelty search; for 700 cars only between random search and BOGP, while there is no statistically significant difference between any two techniques for 800 cars.

In general, we observe that by increasing cars, the average *trip overhead* and *routing cost* of the pareto-optimal configurations across all optimization techniques increase as well. This matches our expectation that with increasing traffic, it is more difficult to optimize the system in absolute terms.

Besides considering each objective individually, we want to investigate how *both objectives together* are satisfied by solutions found by the different optimization techniques. Instead of defining a utility function over the *trip overhead* and *routing cost*, which may introduce some bias toward one objective, we use the well-known quality indicator *hypervolume* [32].[7] In general, the hypervolume measures the volume in the objective space that is dominated by a pareto front. Thus, a higher hypervolume indicates a pareto front of better quality.

Thus, we computed the hypervolume for each pareto front of the 30 replicates for each technique and situation. The resulting data is plotted in Figures 3(g)-(i) and the average and mean hypervolumes are listed in Table III. Since the average and median values do not differ much, we consider the average hypervolume in the following. For the situations with 500 and 700 cars, the pareto-front found by NSGA-II achieves the highest average hypervolume (173.09 and 114.56), followed by random search (172.41 and 113.77), BOGP (171.11 and 111.62), and novelty search (170.44 and 112.91). In contrast, for 800 cars the highest average hypervolume is achieved by the pareto front obtained by random search (85.00), closely followed by novelty search (84.42) and NSGA-II (84.39), and finally BOGP (82.43). Concerning the hypervolume, we notice a statistically significant difference for 500 cars between NSGA-II and BOGP, NSGA-II and novelty search, as well as novelty search and random search; for 700 cars between NSGA-II and BOGP, as well as random search and BOGP; and for 800 cars between random search and BOGP.

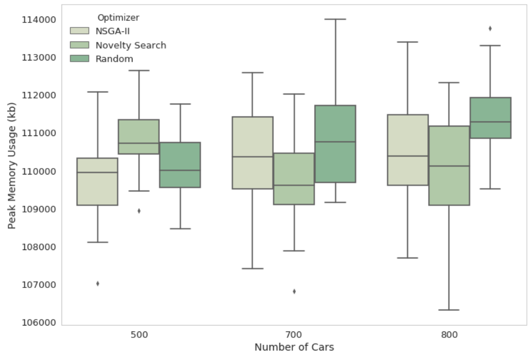
2) Convergence: We evaluate the convergence of the different techniques by evaluating how the hypervolume evolves by plotting the achieved hypervolume over the 100 fitness evaluations for each situation as shown in Figures 3(j)-(l). While for 500 cars, there is no distinct difference between the techniques, we observe for 700 and 800 cars that BOGP achieves slightly better results during the first 10 fitness evaluations than the other techniques. However, later on BOGP converges faster than the other techniques. Thus, BOGP is a promising technique to find good configurations quicker, which supports a faster adaptation cycle while the other techniques may continue their search to find better configurations used for an adaptation later on in time. For instance, one run of NSGA-II with a budget of 1000 fitness evaluations (ten times the budget we considered so far) achieved a hypervolume of 186.14 for 500 cars. This run illustrates that better solutions can be obtained by NSGA-II with a larger budget. However, a budget of 1000 evaluations corresponds to an optimization horizon of 600 minutes (when evaluating 10 configurations in parallel), which prevents a timely adaptation in a traffic system.

3) Overhead: We now discuss the overhead of each optimizer. As such, Figures 4(a) and 4(b) present our performance metrics for 500, 700, and 800 car counts, and Table IV summarizes those results. Note that, for each plot, the optimizers are presented in the order of NSGA-II, novelty search, and random search, respectively. We examine the peak memory overhead (kb) and peak processor usage (%) required to execute RTX.

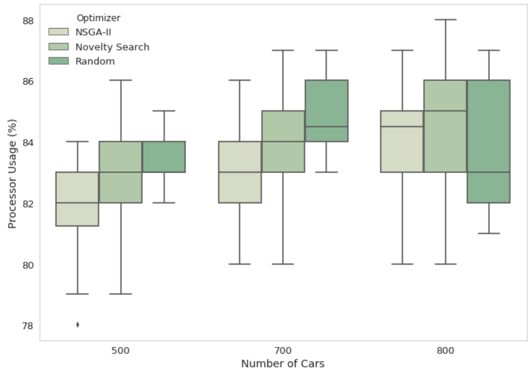
For the presented metrics, a general trend of increasing overhead is seen as the number of cars increases, with random search tending to require the most resources. For both memory overhead and processor usage, we see no statistically significant difference between random, NSGA-II, and novelty search at any of the situations. As such, these results suggest that each of our implemented optimizers, including the baseline, require a similar amount of memory and CPU overhead.

4) Discussion: Given the results from evaluating multiple optimization techniques, we see that these techniques struggle with optimizing the *trip overhead*. Our interpretation of these results is that: (1) We investigated the valuations of the input parameters for the pareto-optimal configurations across all techniques. We found that these valuations are spread in the

[7]We use variant 3 of the hypervolume algorithm by Fonseca et al. [33]. Implementation: https://ls11-www.cs.tu-dortmund.de/rudolph/hypervolume/.



(a) Peak memory usage (kb).



(b) Peak processor usage (%)

Fig. 4. Performance metrics for all evaluations.

TABLE IV
MEMORY OVERHEAD AND PROCESSOR USAGE FOR 500, 700, 800 CARS.

	Technique	Memory Overhead (kb)		Processor Usage (%)	
		Average	Median	Average	Median
500	NSGA-II	109687.2000	109944.0000	82.0333	82.0000
	Novelty Search	110802.1333	110710.0000	82.6667	83.0000
	Random	110135.7333	109994.0000	83.4000	83.0000
700	NSGA-II	110308.6667	110352.0000	83.0333	83.0000
	Novelty Search	109671.6000	109594.0000	84.0333	84.0000
	Random	110873.4667	110750.0000	84.8667	84.5000
800	NSGA-II	110543.6000	110360.0000	84.1667	84.5000
	Novelty Search	109932.2667	110096.0000	84.5000	85.0000
	Random	111425.7333	111274.0000	84.1667	83.0000

search space so that we can assume that there are many local minima. (2) The trip overhead is influenced by all of the seven input parameters, which results in a large search space. (3) The trip overhead is rather noisy (it has high variance). These three aspects make it difficult to optimize the trip overhead. In contrast, the *routing cost* is easier to optimize than the *trip overhead* as it is only affected by one input parameter (*re-routing frequency*). Therefore, a technique might identify and follow a gradient based on the relationship that a higher routing frequency leads to lower routing cost.

Considering the goal of selecting one optimization technique, it depends on which criterion the selection is based on. If it is based on the solution quality, NSGA-II performs best – even though slightly – in two situations (500 and 700 cars) and only slightly worse than the best technique in the remaining situation (800 cars). Nevertheless, random search performs surprisingly well in comparison to the other, more intelligent techniques. A reason for this might be the many local minima that exist for the *trip overhead* (*cf.* previous paragraph) so that a random search may easily catch such a minimum with 100 trials of random configurations. Similar observations, that random search performs well in cases of parameter optimization, have been made [34], [35] and witness that random search can be an effective technique for optimizing high-dimensional, black-box systems. Considering the convergence of the different techniques, BOGP should be selected since it finds good configurations quicker than the other techniques. However, it shows smaller improvement in the solution quality in longer runs. Finally, no selection can be done based on the overhead of the different optimization techniques, since their overhead in terms of memory and processor usage is comparable.

D. Threats to Validity

We have identified the following threats to validity of the evaluation results. First, we have used only a single context parameter to show the feasibility of runtime clustering for situation detection in CrowdNav. We further rely on the well-known k-means clustering algorithm for situation detection, using the Silhouette method for determining the best value of k. Other methods exist to learn the optimal number of clusters, such as XMeans [36], that may lead to different results. Second, when optimizing for a situation, we set the number of vehicles to the largest number in the corresponding cluster, assuming that this is representative of other vehicle numbers in the cluster. Third, we have used the vanilla version of the three optimization strategies we selected. Thus, we did not tune the meta-parameters (e.g., number of generations, crossover rate, etc.) to tailor each technique specifically to CrowdNav. Fourth, this study focused on CrowdNav as a representative of the class of systems corresponding to black-box, high-dimensional, and expensive problems. Therefore, although our approach of planning as optimization may generalize to other systems in this class, the evaluation results are obtained for a specific simulated system (CrowdNav) and cannot be generalized to other systems.

VI. Challenges

We have presented a proof of concept of the planning as optimization approach, together with an empirical study of different optimization techniques applied in a complex system that corresponds to a black-box, high-dimensional, and computationally expensive optimization problem. Our evaluation results indicate that none of the compared techniques is superior in optimizing CrowdNav in terms of solution quality, convergence, and overhead. Moreover, the results indicate that different techniques perform better in different situations of the running system with respect to different objectives. Thus, these insights motivate our vision of *self-learning continuous optimization*: to use multiple optimization techniques at runtime and switch between them according to the situation and objective of optimization, while always having an optimization process and a situation identification process running. To realize the vision, our proposed approach must be extended by addressing the following challenges.

Continuous clustering. While performing clustering at runtime based on system outputs to identify distinct situations, the number and range of situations may evolve in time. For instance, in the first 30 minutes of collecting output data three situations may be identified; this number may evolve to four after 60 minutes. These four situations may even have no overlap with the previous three. An approach for self-learning continuous optimization should be able to detect situations that do not change, or similar situations between consecutive learning phases for which optimal configurations can be reused. Moreover, it should effectively "forget" old data to identify clusters that correspond to the latest environment.

Seamless operation of optimizers. Our planning as optimization approach needs to be able to pause an optimization process when the current situation changes and continue it when the situation arises again. For instance, when the current situation s_a changes to s_b while optimizer o_a is running, o_a needs to store its status (e.g., the best solutions found so far) to the `Knowledge Base` (Figure 1) to reuse it when s_a appears again. Self-learning continuous optimization not only needs to be able to pause and resume the operation of optimizers, but also dynamically switch between optimizers at runtime.

Automated comparison of optimizers. In our empirical study, we have compared three optimizers based on solution quality, convergence, and overhead. We presented all the results and drew conclusions which can be used for choosing one optimizer over another. In self-learning continuous optimization, such conclusions need to be taken by the system itself, which raises a number of challenges: How many iterations to perform per optimizer? How many samples to collect for the evaluation of a configuration? Which criteria to use in the comparison? Consider also that different situations (e.g., accidents) may require a change in the choice of optimizers (e.g., select the fast and less effective optimizer).

VII. Conclusion

In this paper, we presented the *planning as optimization* approach that uses optimization strategies to discover optimal system configurations at runtime for each distinct situation that is dynamically identified at runtime. We instantiated our approach with well-known techniques such as the k-means clustering algorithm to identify distinct situations, and Bayesian optimization with Gaussian Processes (BOGP), NSGA-II, and novelty search for finding optimal configurations. Our approach tackles complex, real-world systems such as CrowdNav that can be modeled as black-box, high-dimensional, and computationally expensive optimization problems.

We show the feasibility of planning as optimization by dynamically identifying distinct situations via clustering and by identifying optimal configurations via optimization techniques. We further compare the solution quality, convergence, and overhead of three optimization techniques in an empirical study with CrowdNav. The results show that no technique is *significantly* superior for all three situations in terms of the solution quality. However, NSGA-II performs slightly better in terms of solution quality in two situations while BOGP converges faster in all three situations. With respect to CPU and memory overhead, no technique is significantly different.

Finally, we discussed our vision of self-learning continuous optimization and related open research challenges: (i) continuous clustering; (ii) seamless operation of optimizers; and (iii) automated comparison of optimizers.

Acknowledgment

This research has been supported in part by NSF grant CNS-1657061, the Michigan Space Grant Consortium, the Comcast Innovation Fund, Google Cloud, and Oakland University. This work is part of the ViM project funded by the Bavarian Ministry of Economic Affairs, Regional Development and Energy. This work has been partly developed in the *FLASH* project funded by the German Science Foundation (DFG).

References

[1] B. H. C. Cheng, R. Lemos, H. Giese, P. Inverardi *et al.*, "Software engineering for self-adaptive systems: A research roadmap," in *Software engineering for self-adaptive systems*. Springer, 2009, pp. 1–26.

[2] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker, "A Survey on Engineering Approaches for Self-Adaptive Systems," *Pervasive and Mobile Computing Journal*, vol. 17, pp. 184–206, 2015.

[3] A. Rodrigues, R. Caldas, G. N. Rodrigues, T. Vogel, and P. Pelliccione, "A learning approach to enhance assurances for real-time self-adaptive systems," in *Proc. SEAMS*, 2018, pp. 206–216.

[4] B. H. C. Cheng, P. Sawyer, N. Bencomo, and J. Whittle, "A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty," in *Proc. MODELS*, 2009, pp. 468–483.

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

[6] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic qos management and optimization in service-based systems," *IEEE Trans. Softw. Eng.*, vol. 37, no. 3, pp. 387–409, 2011.

[7] S. Shan and G. G. Wang, "Survey of modeling and optimization strategies to solve high-dimensional design problems with computationally-expensive black-box functions," *Structural and Multidisciplinary Optimization*, vol. 41, no. 2, pp. 219–241, 2010.

[8] D. Weyns, "Software engineering of self-adaptive systems," in *Handbook of Software Engineering*. Springer, 2019, pp. 399–443.

[9] B. Porter and R. R. Filho, "Losing Control: The Case for Emergent Software Systems Using Autonomous Assembly, Perception, and Learning," in *Proc. SASO*, 2016, pp. 40–49.

[10] P. Pilgerstorfer and E. Pournaras, "Self-adaptive learning in decentralized combinatorial optimization: A design paradigm for sharing economies," in *Proc. SEAMS*. IEEE, 2017, pp. 54–64.

[11] J. Jiang, S. Sun, V. Sekar, and H. Zhang, "Pytheas: Enabling Data-Driven Quality of Experience Optimization Using Group-Based Exploration-Exploitation," in *Proc. NSDI*, 2017, pp. 393–406.

[12] S. Schmid, I. Gerostathopoulos, C. Prehofer, and T. Bures, "Self-adaptation based on big data analytics: A model problem and tool," in *Proc. SEAMS*, 2017, pp. 102–108.

[13] D. Wolpert and W. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, 1997.

[14] C. Kinneer, Z. Coker, J. Wang, D. Garlan, and C. L. Goues, "Managing uncertainty in self-adaptive systems with plan reuse and stochastic search," in *Proc. SEAMS*, 2018, pp. 40–50.

[15] B. Bischl, J. Richter, J. Bossek, D. Horn, J. Thomas, and M. Lang, "mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions," 2018, arXiv:1703.03373.

[16] G. A. Moreno, A. V. Papadopoulos, K. Angelopoulos, J. Cámara, and B. Schmerl, "Comparing model-based predictive approaches to self-adaptation: Cobra and pla," in *Proc. SEAMS*, 2017, pp. 42–53.

[17] E. Fredericks, K. Bowers, K. Price, and R. Hariri, "CAL: A smart home environment for monitoring cognitive decline," in *Proc. ICDCS*, 2018.

[18] N. Bencomo and A. Belaggoun, "Supporting decision-making for self-adaptive systems: from goal models to dynamic decision networks," in *Proc. REFSQ*, 2013, pp. 221–236.

[19] S. Tomforde, H. Prothmann, F. Rochner, J. Branke, J. Hähner, C. Müller-Schloer, and H. Schmeck, "Decentralised Progressive Signal Systems for Organic Traffic Control," in *Proc. SASO*, 2008, pp. 413–422.

[20] J. Wuttke, Y. Brun, A. Gorla, and J. Ramaswamy, "Traffic Routing for Evaluating Self-Adaptation," in *Proc. SEAMS*, 2012, pp. 27–32.

[21] S. Na, L. Xumin, and G. Yong, "Research on k-means Clustering Algorithm: An Improved k-means Clustering Algorithm," in *Proc. ITSS*, 2010, pp. 63–67.

[22] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987.

[23] A. Starczewski and A. Krzyżak, "Performance Evaluation of the Silhouette Index," in *Artificial Intelligence and Soft Computing*. Springer, 2015, pp. 49–58.

[24] I. Gerostathopoulos, C. Prehofer, and T. Bures, "Adapting a system with noisy outputs with statistical guarantees," in *Proc. SEAMS*, 2018, pp. 58–68.

[25] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. d. Freitas, "Taking the Human Out of the Loop: A Review of Bayesian Optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.

[26] D. Horn, T. Wagner, D. Biermann, C. Weihs, and B. Bischl, "Model-Based Multi-objective Optimization: Taxonomy, Multi-Point Proposal, Toolbox and Benchmark," in *Evolutionary Multi-Criterion Optimization*. Springer, 2015, vol. 9018, pp. 64–78.

[27] W. Ponweiser, T. Wagner, D. Biermann, and M. Vincze, "Multiobjective optimization on a limited budget of evaluations using model-assisted $\mathcal{S}$-metric selection," in *Proc. PPSN*, 2008, pp. 784–794.

[28] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, 2002.

[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, 2012.

[30] J. Lehman and K. O. Stanley, "Exploiting open-endedness to solve problems through the search for novelty," in *Proc. ALIFE*, 2008.

[31] A. Arcuri and L. Briand, "A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Software Testing, Verification and Reliability*, vol. 24, no. 3, pp. 219–250, 2014.

[32] S. Wang, S. Ali, T. Yue, Y. Li, and M. Liaaen, "A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering," in *Proc. ICSE*, 2016, pp. 631–642.

[33] C. M. Fonseca, L. Paquete, and M. Lopez-Ibanez, "An improved dimension-sweep algorithm for the hypervolume indicator," in *Intl. Conference on Evolutionary Computation*. IEEE, 2006, pp. 1157–1163.

[34] K. Seymour, H. You, and J. Dongarra, "A comparison of search heuristics for empirical code optimization," in *Proc. Cluster*, 2008, pp. 421–429.

[35] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, 2012.

[36] D. Pelleg and A. Moore, "X-means: Extending k-means with efficient estimation of the number of clusters," in *Proc. ICML*, 2000, pp. 727–734.