Traffic routing for evaluating self-adaptation

Article · June 2012

DOI: 10.1109/SEAMS.2012.6224388

CITATIONS
18

READS
123

4 authors, including:

Jochen Wuttke
University of Washington Seattle
19 PUBLICATIONS **889** CITATIONS
SEE PROFILE

Alessandra Gorla
Universität des Saarlandes
44 PUBLICATIONS **2,120** CITATIONS
SEE PROFILE

Traffic Routing for Evaluating Self-Adaptation

University of Washington Technical Report UW-CSE-12-03-04

Jochen Wuttke †, Yuriy Brun †, Alessandra Gorla⋆, Jonathan Ramaswamy †

† University of Washington
Seattle, WA, USA
{wuttke, brun, ramasj}@cs.washington.edu

⋆ University of Lugano
Lugano, Switzerland
alessandra.gorla@usi.ch

***Abstract*—Toward improving the ability to evaluate self-adaptation mechanisms, we present the automated traffic routing problem. This problem involves large numbers of agents, partial knowledge, and uncertainty, making it well-suited to be solved using many, distinct self-adaptation mechanisms. The well-defined nature of the problem allows for comparison and proper evaluation of the underlying mechanisms and the involved algorithms. We (1) define the problem, (2) outline the sources of uncertainty and partial information that can be addressed by self-adaptation, (3) enumerate the dimensions along which self-adaptive systems should be evaluated to provide a benchmark for comparison of self-adaptation and traditional mechanisms, (4) present ADASIM, an open-source traffic routing simulator that allows easy implementation and comparison of systems solving the automated traffic routing problem, and (5) demonstrate ADASIM by implementing two traffic routing systems.**

***Keywords*-adaptive routing, benchmark, exemplar**

I. INTRODUCTION

Self-adaptation is one approach to handling the growing complexity of software systems [4], [9]. Self-adaptive systems observe their environments and determine when something prevents them from meeting their requirements. The systems then adapt to satisfy those requirements.

Self-adaptation is a popular area of research, e.g., [3], [7], [8], [14], [15], [25]. Yet, the lack of a high-level organization of the research and a standardized framework for comparing techniques and evaluating approaches has been hurting progress. Further, the adoption of self-adaptation has been hindered by poor understanding of how self-adaptation techniques compare to other, more traditional approaches.

We, as well as others, have previously argued that to facilitate comparison, it is helpful to produce benchmarks with clearly identified evaluation dimensions [2], [4], [6]. In this paper, we present one such benchmark, the *automated traffic routing problem*. This problem deals with traveling vehicles on a constrained set of roadways. Each vehicle has a starting and an ending location and a time at which it starts its journey. On each road, speed limits and traffic affect how fast cars may travel. Some events, such as accidents, or construction, can unexpectedly affect this speed and even close some roads. There are numerous quantities and qualities that a traffic routing system may choose to optimize, such as the total travel time, the worst-case vehicle's travel time, or the environmental pollution. Because the automated traffic routing problem involves large numbers of agents, partial knowledge, and uncertainty, it is well-suited to be solved using self-adaptation mechanisms.

The systems that solve the automated traffic routing problem can be non-adaptive, centralized self-adaptive, or decentralized self-organizing. The techniques can be evaluated on scalability, robustness to noise in observation, robustness to vehicle and goal churn, answer quality, and resource consumption. Some existing research has tackled related problems, e.g., [10], [15]. However, without a formal, standardized, and consistent problem statement and clear evaluation dimensions, comparing these techniques has been nearly impossible.

This paper addresses this shortcoming. Section II presents such a problem statement for the automated traffic routing problem. Section III outlines the sources of uncertainty and partial information. Section IV provides a benchmark for self-adaptation and traditional mechanisms by identifying the automated traffic routing problem's evaluation dimensions. Section V describes ADASIM, an open-source traffic routing simulator that allows easy implementation and comparison of systems solving the automated traffic routing problem and Section VI demonstrates our experience using ADASIM to implement, simulate, and compare techniques. Section VII places our work in the context of related research. Finally, Section VIII summarizes our contributions.

II. THE AUTOMATED TRAFFIC ROUTING PROBLEM

In this section, we first describe the automated traffic routing problem informally. This description is sufficient to understand the problem, the inherent uncertainty and dimensions along which techniques can be evaluated, and our simulator. We then, for completeness, formalize the definitions. The formalism can give a deeper understanding of the problem and resolve ambiguities that often result from informal descriptions.

A. Intuitive Description

The automated traffic routing problem involves vehicles, traveling on a map. Each vehicle has a starting and a target point and a starting time. The goal of each vehicle is to traverse the map, along its streets, from the starting to the

target point. The vehicles compete for resources. The streets impose speed limits, and as the number of vehicles and congestion on a given street increases, the maximum allowed speed decreases. Some events, such as traffic accidents or road closures, can cause further slowdowns.

Solving the automated traffic routing problem requires decision making. The vehicles themselves are a natural set of decision-making agents, but it is also possible for the streets, intersections, and other virtual agents to make decisions. It is possible for the agents to have a complete and accurate view of the world, though it is more realistic for each agent to only be able to observe a small, local part of the world. Further, such observations are likely to be noisy and create uncertainty. As we describe in Section III, these properties and large instance sizes make this problem ideal to be solved with self-adaptation.

Systems that solve the automated traffic routing problem will differ in two important ways: First, their *answers* to instances of the automated traffic routing problem differ. For example, with reasonable assumptions, a centralized solution may be able to always find the answer that minimizes the sum of the travel times of all the vehicles. In contrast, a decentralized solution is unlikely to always minimize this quantity. Second, the characteristics of the *approach* the solutions take to find the answers differ. For example, a centralized solution may require knowing all vehicles' goals ahead of time, perform the bulk of its computation before movement can begin, and contain a single point of failure. In contrast, a decentralized solution may require no a priori knowledge, allow vehicles to start moving as soon as they wish, and contain no single point of failure. Further, a decentralized solution likely has lower communication costs and scales better with the number of vehicles.

B. Formal Definitions

We now formalize the automated traffic routing problem.

An instance of the automated traffic routing problem consists of: (1) a node-weighted, directed graph, with a monotonically non-decreasing, non-negative traffic delay function associated with every node, and (2) a set of vehicles, each with a starting and an ending node and a starting time.

Definition 1 (Automated Traffic Routing Problem Instance)**.** Let N be a set of nodes. Let C be a set of directed connections between nodes $c \in N^2$. Let $w\colon N \to \mathbb{Z}_{\geq 0}$ map nodes to non-negative integer weights. Then $G = \langle N, C, w \rangle$ is a node-weighted, directed graph.

Let F be the set of all $f\colon \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ monotonically non-decreasing, non-negative functions. Then let $d\colon N \to F$ map each node to one such function.

Let V be a set of vehicles, where each vehicle $v = \langle s_v \in N, f_v \in N, t_v \in \mathbb{Z}_{\geq 0} \rangle$ is a triple of a starting and ending vertex in G and a starting time.

Then an automated traffic routing problem instance is the triple $\mathcal{P} = \langle G, d, V \rangle$.

An answer to an instance of the automated traffic routing problem is, for each vehicle, a path along the graph that will take that vehicle from its starting to its ending vertex.

Definition 2 (Automated Traffic Routing Answer)**.** Let $\mathcal{P} = \langle G, d, V \rangle$ be an instance of the automated traffic routing problem. For all $z \in \mathbb{Z}_{\geq 0}$, for each $v \in V$, let $a_v = \langle n_1, n_2, \ldots, n_z \rangle$, be such that:

- each n_i is a node in G,
- the sequence $\langle n_1, n_2, \ldots, n_z \rangle$ forms a valid path within G,
- $n_1 = s_v$, and
- $n_z = f_v$.

Let $\mathcal{A}$ be a set of a_v, one for each $v \in V$. Then $\mathcal{A}$ is an answer to $\mathcal{P}$.

A measurement uncertainty filter defines how each entity's properties' values are distorted when observed.

Definition 3 (Measurement Uncertainty Filter)**.** Let E be a finite alphabet of entities. Let V be a finite alphabet of values that properties can take on. For all $e \in E$, let Pr_e be the set of e's properties. Then for each $pr_e \in Pr_e$, a measurement uncertainty filter $m_{pr_e}\colon V \to V \cup \{\text{␣}\}$, where (␣ is a special *blank* symbol).

A data privacy policy defines which entity's properties which agents can access.

Definition 4 (Data Privacy Policy)**.** Let A be a finite alphabet of agents. Let E be a finite alphabet of entities. Let V be a finite alphabet of values that properties can take on. For all $e \in E$, let Pr_e be the set of e's properties. Then for each $a \in A$, for each $pr_e \in Pr_e$, a data privacy policy $r_{a,pr_e}\colon V \to V \cup \{\text{␣}\}$, where (␣ is a special *blank* symbol).

Finally, an automated traffic routing problem solution produces answers to problem instances. A solution consists of: (1) a set of sensors (measurement uncertainty filter and data privacy policy pairs), and (2) a set of agents and agent algorithms such that each vehicle is assigned to exactly one agent.

Definition 5 (Automated Traffic Routing Solution)**.** Let A be a finite alphabet of agents. Let E be a finite alphabet of entities. Let Pr be the set of all possible properties of all possible entities. Let $\hat{\mathcal{P}}$ be the set of all possible automated traffic routing problem instances. Then for each $\mathcal{P} \in \hat{\mathcal{P}}$, let $M_{\mathcal{P}}$ map $pr \in Pr$ to a measurement uncertainty filter. And for each $\mathcal{P} \in \hat{\mathcal{P}}$, for each agent $a \in A$, let $R_{\mathcal{P}}$ map each pair $\langle a, pr \in Pr \rangle$ to data privacy policy.

Let $\hat{V}$ be the set of all possible vehicles. Then let $\hat{a}\colon \hat{V} \to A$ assign each vehicle to an agent.

Let $\hat{\mathcal{A}}$ be the set of all possible automated traffic routing answers. Then let $\mathcal{S}\colon \hat{\mathcal{P}} \to \hat{\mathcal{A}}$ such that $\mathcal{S}(\mathcal{P}) = \mathcal{A} \Rightarrow \mathcal{A}$ is a valid answer for $\mathcal{P}$, and let $\bar{\mathcal{S}}$ be the set of algorithms, one per agent, that execute to compute $\mathcal{S}$.

Then, if V is the set of vehicles in $\mathcal{P}$, and if $\bar{\mathcal{S}}$ operates with

the set of agents $\hat{A} \subseteq A$, and if each agent $a \in \hat{A}$ only operates on vehicles v if $\hat{a}(v) = a$, and if for all entities e, for each of e's properties pr_e, for all values v of pr_e, in the process of $\bar{\mathcal{S}}$'s execution, the only information a accesses about e is $(M_{\mathcal{P}}(pr_e))((R_{\mathcal{P}}(a, pr_e))(v))$, then $\bar{\mathcal{S}}$ is a set of algorithms that make up the automated traffic routing solution $\mathcal{S}$.

III. Applicability to Self-Adaptation

The automated traffic routing problem is well suited for some self-adaptation techniques for three reasons: (1) it involves a large number of agents, (2) each agent has access only to partial information, and (3) each agent's view may involve significant uncertainty.

Problems with a small number of agents lend themselves to centralized, static solutions because collecting and managing information about all the agents is feasible. In contrast, the automated traffic routing problem can involve numerous vehicles, streets, and intersections. Solutions that realistically represent the real world of vehicular traffic must take into account changing goals at runtime (e.g., drivers may suddenly decide to change their destinations), and locally-made decisions (e.g., drivers select their own routes, without notifying a centralized agent).

Problems for which it is possible for a single agent to have complete knowledge of the environment again lend themselves to centralized, static solutions. In contrast, the decision-making agents in most realizations of the automated traffic routing problem, such as the vehicles, streets, and intersections, have only a partial view of the world. For example, a driver may only see the traffic conditions on its currently-traveled street. The partial views may cause agents to make globally-suboptimal choices and require their strategies to change at runtime, as information changes or becomes available to the agents.

Finally, problems with exact environmental descriptions that can be sensed precisely also lend themselves to centralized, static solutions that do not need to adapt at runtime. In contrast, the automated traffic routing problem has numerous sources of uncertainty that may cause agents to make suboptimal decisions that need to be changed later. For example, a vehicle's sensors may be noisy, and knowledge among agents may be inconsistent. Additionally, sudden environmental changes, such as accidents, can change traffic and other conditions unpredictably, invalidating some of the vehicles' knowledge. The ADASIM simulator we describe in Section V allows for explicit modeling of each of these kinds of uncertainty.

IV. Solution Comparison Dimensions

There are many paths each vehicle in an instance of the automated traffic routing problem could follow. Some solutions may select these paths once and never change them. Other solutions may adapt these paths as traffic conditions change. Because solutions can employ such drastically different approaches, comparing them is a difficult task. However, without careful, methodical comparison, research cannot make steady progress [2]. In this section, we describe several dimensions along which solutions to the automated traffic routing problem should be evaluated and compared. While this list is not exhaustive, we try to identify the most important dimensions.

Answer Quality: The answers to each problem instance can vary in quality. Vehicles may reach their goals quickly or be delayed by traffic or long routes. As solutions generate answers, they may aim to optimize different measures. For example, solutions may choose to minimize the sum of all vehicles' travel times, the maximum time a vehicle travels, etc. Since producing answers is the main goal of the automated traffic routing problem solutions, this is an important comparison dimension.

Scalability: The automated traffic routing problem solutions need to scale well in two dimensions: the number of vehicles and the size of the map. For each dimension, it is important to consider the growth of the computation and communication costs. Both the worst-case asymptotic complexity and the typical-case in practice analyses are appropriate. Simulation or experimentation may assist with the latter analysis. Further, it is important to consider where the computation takes place. For example, a single, centralized agent may be powerful enough to execute complex algorithms far faster than the vehicles themselves.

Robustness to Sensor Uncertainty: As the agents observe their environment, the data their sensors return about other vehicles' speed and street congestion are noisy. A solution that is robust to noise does not produce vastly different answers when experiencing small sensor reading differences. If a small perturbation results in a large change to the answer, then the solutions will likely be ineffective at dealing with noisy, dynamically-changing, real-world data.

Robustness to Agent and Street Failure: Vehicles and other agents may fail at any time. A solution that is robust to agent failures can withstand such failures with minimal effects on the answer it produces. For example, if one vehicle fails, a robust solution would not negatively affect other vehicles. Further, robust solutions would adjust the paths of other vehicles once failures occur if those paths can be improved given the failures.

Streets can also fail and become closed. Some such failures can prevent certain vehicles from reaching their destinations, while only delaying others. A solution that is robust to street failures can react, at runtime, to such failures and reroute vehicles as necessary.

Robustness to Churn: The automated traffic routing problem allows for vehicles to start and finish at different times. A solution that is robust to churn does not require knowing about vehicles until their starting times and can accommodate new vehicles that choose to join. Further, the

answers such a solution produces are not greatly affected by vehicles deciding to halt their travel in the middle of their path or change their destination.

Resource Consumption: Agents consume resources. For example, the computation and communication the agents perform cost energy. Thus, solutions can be compared based on their resource consumption. Further, solution resource consumption can be compared to the minimal amount of resources required to solve the problem.

V. The ADASIM Simulator

We have built the ADASIM simulator for evaluating and comparing automated traffic routing problem solutions (http://adasim.googlecode.com). ADASIM is written in Java and is configurable and extendable, as we discuss in Section V-B. ADASIM makes up a significant part of the contribution of this paper as eases the comparison of the automated traffic routing problem solutions along the dimensions outlined in Section IV. ADASIM implements most of the features we described in Section II, though it is an ongoing project with new features being added regularly. We encourage ADASIM users to submit feature requests and, of course, bug reports, via http://adasim.googlecode.com. Since ADASIM is open-source, we also welcome code contributions for specific issues in the project's issue tracker.

The remainder of this section will first overview ADASIM's high-level design and then describe how to use ADASIM to evaluate solutions. Then, Section VI will discuss our experience using ADASIM to evaluate two specific solutions.

A. ADASIM *Design*

ADASIM is a discrete event simulator that tracks vehicles traveling on a map. Each vehicle has a starting and an ending point. The speed of a vehicle traveling on a street depends on that street's traffic delay function and the number of other vehicles on that street. Since the simulation is discrete, during each simulation cycle, each vehicle gets a chance to perform an action. Vehicles can move along a street, move to a connecting street at an intersection, or stop. All notions of time in the model are discrete as well. The travel time is measured in cycles, and all measures of distance, speed, and traffic delay are expressed in terms of cycles. For example, a street may take 10 cycles to travel if no other vehicles are present, but the travel time may increase to 100 cycles if 100 other vehicles are traveling the same street at the same time.

ADASIM has six kinds of abstract entities: (1) a map, (2) vehicles, (3) agents that make routing decisions and collect and store information, (4) sensors that allow agents to observe the environment, (5) measurement uncertainty filters that control the noise and other sources of uncertainty in the sensor measurements, and (6) data privacy policies that allow vehicles and streets to restrict part or all information about themselves from sensors.

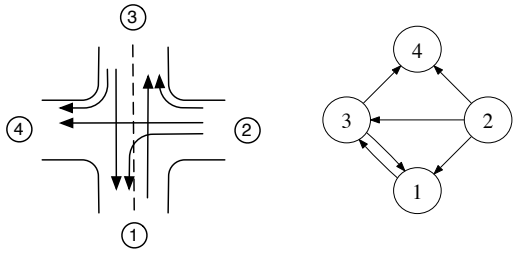


Figure 1. An example map of an intersection (a) and the corresponding ADASIM graph.

The *map* is represented by a node-weighted graph. The nodes in the graph represent street segments, and the edges represent connections between segments. There are several advantages to this representation of street segments as nodes. For example, it enables modeling intersections with restrictions on which turns vehicles can make (e.g., a disallowed left turn). Figure 1(a) shows a sample intersection of four street segments. The arrows illustrate the six legal travel directions in this intersection. Figure 1(b) shows the ADASIM graph that represents this intersection. Note that the graph has six edges, one for each legal travel direction in the intersection.

Each *vehicle* starts traveling at its starting simulation cycle from its starting street segment and aims to reach its target street segment. Each vehicle has associated with it an agent responsible for deciding that vehicle's next action. Vehicles can sense the environment, as we describe below. Once a vehicle reaches its target, it is removed from the simulation. The vehicle's agent may also decide to remove the vehicle.

The ADASIM *agents* make routing decisions on behalf of the vehicles. Each agent has associated with it a routing algorithm that it uses to decide what its vehicles will do next.

Sensors allow agents to observe the environment. Each sensor is associated with a single entity, such as an agent, a vehicle, or a street segment. The sensor's entity registers with the sensor what quantities may be sensed about that entity. For example, a vehicle may allow its sensor to share its speed and current location, but not its starting and target street segments. Other agents can query the sensors for this information. Before revealing the queried information, sensors apply two filters: one that injects measurement uncertainty and one that preserves the underlying entity's privacy.

Each entity specifies the *measurement uncertainty filter* that is applied to each of its sensed pieces of information. For example, a filter may add 10% Gaussian noise to the speed of a vehicle, or return a set of neighboring street segments instead of a single current position.

Finally, each entity specifies the *data privacy policy* that sensors apply to its information before sharing it. For example, vehicles may be willing to share their location with intersection agents but not with other vehicles' agents.

The current implementation of ADASIM places some lim-

itations on the features we describe above. We made these simplifications to limit the scope of ADASIM's supported features and improve the quality of the current release. The current ADASIM implementation allows only for vehicles themselves to act as agents, and there are some limits on the diversity of measurement uncertainty filters. Since data privacy filters use the same mechanism, they are subject to the same limitations. We plan to add the full set of features in the next release. In the future, we envision that agents can be positioned anywhere: on vehicles; on intersections; on street segments; on collections of vehicles, intersections, and street segments; or virtually, without a physical location.

B. Using and Extending ADASIM

ADASIM simulates the execution of an automated traffic routing problem solution on an automated traffic routing problem instance. Thus, to execute an ADASIM simulation, one must specify the solution and the problem instance. Doing so consists of defining the necessary concrete instances of the six abstract entities defined in Section V-A. The map, vehicle set, measurement uncertainty filters, and data privacy policies define the instance of the problem. The agents and sensors define the solution.

The user specifies the problem instance and solution using an ADASIM XML configuration file (see http://code.google.com/p/adasim/wiki/XML_config). Each street segment's traffic delay function, each agent's routing algorithm, each measurement uncertainty filter, and each data privacy policy is specified via the name of a Java class. Several such classes are built into the ADASIM release. For example, ADASIM ships with traffic delay functions that increase the travel time on a street segment linearly and quadratically with the number of traveling vehicles once that number passes a threshold. In Section VI, we will discuss several routing algorithms also built into ADASIM.

For other functions, algorithms, filters, and policies, the user writes a new Java class (implementing the appropriate Java interface) and specifies that class in the configuration. This allows specifying arbitrary routing algorithms and restricting the problem with traffic load, sensor noise, and privacy constraints. Thus, ADASIM can simulate and evaluate a vast range of automated traffic routing problem solutions.

Once ADASIM loads the solution and problem instance, it simulates the discrete event execution of the vehicles traveling around the map. Whenever a vehicle arrives at its destination, attempts to make an illegal transition on the map, or is told to stop by its agent, that vehicle is removed from the simulation. The simulation halts when no more vehicles remain. Since it is possible for vehicles to enter infinite loops, the user can halt the simulation at any time.

ADASIM logs all vehicle events that take place. These logs contain each vehicle's path, the conditions each agent encountered when it made decisions, and the decisions themselves. Of course, ADASIM also logs illegal actions and vehicle removal. From the logs, it is possible to reconstruct the travel time for each vehicle as well as congestion information at each node during each simulation cycle. The ADASIM release includes several scripts for parsing log files, documented at http://code.google.com/p/adasim/wiki/Scripts.

VI. ADASIM EXPERIENCE

To demonstrate the ease of using ADASIM to compare automated traffic routing problem solutions, we implemented two such solutions and evaluated them along the answer quality dimension. The first solution uses a non-adaptive technique while the second self-adapts to traffic conditions. In this experience, we discovered several surprising results about the seemingly simple solutions that would have been hard to identify and understand without a simulation or a prototype deployment. For example, whereas we expected that with heavy traffic, the self-adaptive solution would significantly outperform the solution that ignored traffic, we found that properties of the map on which the vehicles were traveling significantly affected the benefits of self-adaptation. Further, we found that the self-adaptive solution can cause unintended and undesired behavior, such as livelock.

The goal of this section is not to exhaustively demonstrate ADASIM's features, but only to show the ease of (1) creating new solutions and (2) evaluating and comparing them.

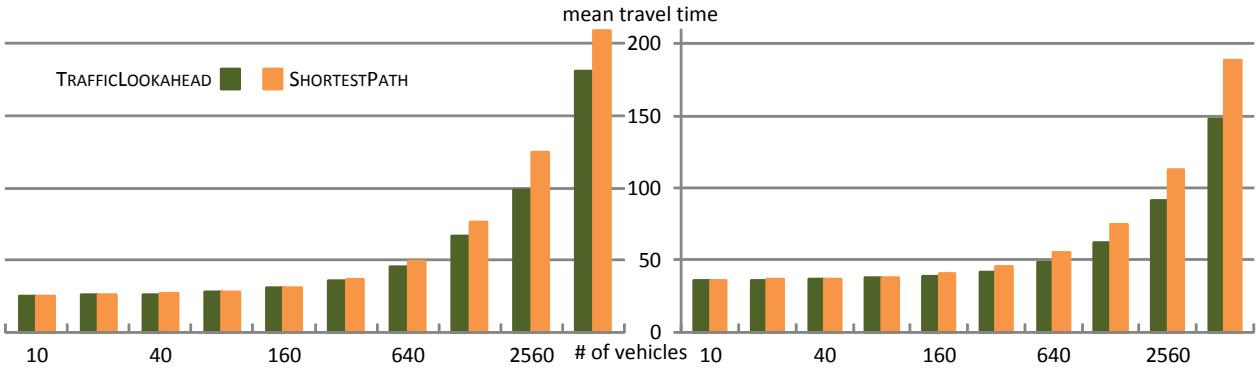
A. Solutions and Problem Instances

We implemented two solutions to the automated traffic routing problem. The first, called SHORTESTPATH, is non-adaptive. The second, called TRAFFICLOOKAHEAD, adapts to traffic conditions.

In SHORTESTPATH, each vehicle has its own agent. The agent uses Dijkstra's algorithm to find the shortest path from the vehicle's start to its destination without taking into account the traffic conditions. The vehicle then follows that path without making adjustments.

In TRAFFICLOOKAHEAD, each vehicle also has its own agent. Each agent can see the traffic conditions at a fixed radius of street segments around its vehicle. The agent also uses Dijkstra's algorithm to compute the fastest path to the destination given its partial view of the traffic conditions and the vehicle begins following that path. After traveling through a street segment, the agent updates its traffic information and recomputes the path from the current location. The vehicle then adjusts its travel accordingly.

We ran our experiments on two types of maps, each having 168 street segments. The first type, called *random*, is randomly-generated. For each ordered pair of street segments, we connected those segments with the probability that led to each segment having, on average, four connections. We chose four because most real-life intersections are four-way. We chose each street segment's length uniformly randomly from $\{3, 4, 5, 6\}$ and the time to travel each segment increases linearly with the number of cars on that segment.



(a) *random* map (b) *city* map

Figure 2. A comparison of the TRAFFICLOOKAHEAD and the SHORTESTPATH solutions on the *random* (a) and *city* (b) maps. In heavy traffic, on *random* maps, SHORTESTPATH vehicles travel 15% longer than TRAFFICLOOKAHEAD vehicles. In contrast, in heavy traffic, on *city* maps, SHORTESTPATH vehicles travel 28% longer than TRAFFICLOOKAHEAD vehicles.

The second type of map, called *city*, is a more realistic representation of a real city map. Figure 3 graphically depicts the *city* map, a regular 6×6 block grid made up of seven horizontal and seven vertical streets (resulting in 168 directed, or one-way, street segments). At each intersection, each street segment connects to every other street segment at this intersection, and additionally allows turning around. Each segment has a length of 5. The 2 six-segment streets that divide the *city* map into four equal quadrants (thicker lines in Figure 3) represent highways that can handle 1000 cars without being affected by traffic. After 1000, the number of cycles necessary to traverse each segment increases linearly with each new vehicle. All other street segments have the same traffic delay functions as the *random* map.

B. Experimental Setup

To explore how traffic conditions impact our two solutions, we produced problem instances with $10 \cdot 2^i$ vehicles, for each $i \in \{0,1,2,3,\ldots,9\}$. We chose each vehicle's starting and ending points uniformly randomly from the map's street segments. We allowed each TRAFFICLOOKAHEAD agent to see the traffic conditions up to five street segments away from the agent's location. The agents recomputed the vehicles' paths at every intersection. We did not model measurement noise for these experiments. We repeated the experiments on the *random* and *city* maps, and ran each experiment enough times to average the travel times of 10,000 vehicles.

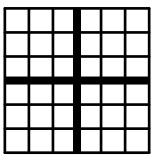


Figure 3. A *city* map is a regular, 6×6 block grid with seven horizontal and seven vertical streets. Each street segment has the same length but the bold segments represent highways and have a significantly higher tolerance for traffic.

C. Results and Discussion

Figure 2 shows the average travel time for the SHORTESTPATH and TRAFFICLOOKAHEAD solutions on *random* and *city* maps. Our simulations revealed several counterintuitive and surprising results. First, TRAFFICLOOKAHEAD only slightly outperforms SHORTESTPATH on the *random* maps. On *random* maps, in heavy traffic, SHORTESTPATH vehicles travel 15% longer than TRAFFICLOOKAHEAD vehicles. On *city* maps the difference is 28%. This finding indicates that the presence of highways can effectively relieve traffic load, whereas the same number of non-highway streets cannot. Second, in a simulation with half the vehicles using SHORTESTPATH and half using TRAFFICLOOKAHEAD, the difference in vehicle travel times was small (data not shown). This finding indicates that while TRAFFICLOOKAHEAD vehicles can relieve traffic load, both types of vehicles benefit from such relief. Third, TRAFFICLOOKAHEAD can sometimes cause livelock, a significant problem from which SHORTESTPATH does not suffer (data not shown).

ADASIM allows for directly comparing SHORTESTPATH and TRAFFICLOOKAHEAD because both solutions execute within the same framework and under the same conditions. This allows for more-precise comparisons than through deriving measures across different evaluation platforms. Here, ADASIM allowed us to quickly and easily identify and understand situations in which one solution's benefits were diminished, despite our earlier intuition.

The total effort required to perform these comparisons using ADASIM is rather small. We had to write one Java class totaling 134 non-comment lines of code. We also had to create scripts to generate the configuration files and maps,

totaling 60 lines of Python code. These scripts and maps, of course, are reusable for future experiments, both on evaluating new solutions, and on comparing those solutions to SHORTESTPATH and TRAFFICLOOKAHEAD.

VII. RELATED WORK

In this paper, we propose a benchmark problem, and a simulator for solutions to that problem, to help evaluate and compare self-adaptive algorithms. Other, more mature research fields have well-established benchmarks to evaluate new techniques against the state-of-the-art. For instance, the software testing and analysis community has the Siemens benchmark suite [16], and the SIR [17] and iBugs [5] repositories, and the programming languages community has DaCapo [1] and SPEC [18] as popular benchmarks. These benchmarks could be adopted for evaluating self-adaptive algorithms along some of the comparison dimensions we discuss in Section IV. For example, the fault data could be used to compare the answer quality of techniques that attempt to heal or avoid functional failures [3], and the performance benchmarks could be used to evaluate the performance and scalability of the techniques. However, to properly evaluate self-adaptive solutions, we need benchmarks that lend themselves specifically to such solutions.

To the best of our knowledge, ZNN.com [26] is the only other attempt to define a common benchmark for self-adaptive techniques, and thus it is the most closely related work to the automated traffic routing problem. ZNN.com is a news content service that serves multimedia content to its customers. ZNN.com has requirements on the timeliness of query responses and operating costs, while environmental conditions, such as spikes in user load, create a need for self-adaptation. Two of ZNN.com's possible adaptations are changing the server pool size and switching between multimedia and text-only content. While ZNN.com is well-suited for systems that make a limited number of pre-specified adaptation decisions, it severely restricts the possibility for creating new solutions with new adaptation mechanisms. In contrast, the automated traffic routing problem offers a wide range of environmental conditions that can influence self-adaptation, and also allows more parts of the overall system to be adapted. This, in turn, creates a large decision space with room for innovative solutions. ZNN.com and the automated traffic routing problem complement each other, allowing for more diversity in the types of self-adaptive techniques one can evaluate.

Frameworks that allow building multiple self-adaptive solutions can also aid the evaluation of such solutions. Villegas et al. survey several self-adaptive techniques and provide a framework that considers the adaptation goals and quality attributes that have to be satisfied by the controller and the monitored system [19]. However, this framework does not lead to empirical *evaluation* of how well techniques meet their goals. In contrast, in this paper, we have proposed a challenging problem that lends itself to self-adaptive solutions, and a simulator to empirically evaluate and compare those solutions.

Simulators are a common way to aid evaluation. There are several network and wireless sensor simulators, such as OMNet++ [12], NS-3 [11], and QualNet [13]. These simulators are widely used among researchers in the networking community and are highly customizable, but deliver only the low-level abstractions of networks. In contrast, ADASIM, is less customizable with regard to the underlying network model, but includes traffic-related abstractions, making implementing solutions to the automated traffic routing problem much easier than with existing network simulators.

Finally, route finding and adaptive traffic control are widely popular research areas. For example, the MACODO middleware for context-driven dynamic agent organizations has been used for routing traffic via traffic light agents [23]. The particular version of the routing traffic problem [23] has also been used elsewhere [22], [21], [24], [20]. Meanwhile, another approach uses self-adaptive cruise control to adjust vehicle speeds based on traffic conditions [15]. These approaches could both be implemented, simulated, and compared using ADASIM. Further, ADASIM allows the direct exploration of the effects of combining the approaches, perhaps learning that they complement or interfere with one another. Instead, today, researchers build their own custom simulators and define their problems independently, making comparison difficult [15], [23].

VIII. CONTRIBUTIONS

As a step toward improving the evaluation of research into self-adaptive systems, we have presented the automated traffic routing problem, an exemplar problem that lends itself to a variety of self-adaptive solutions. Our goal is to provide a canonical problem to allow (1) researchers to evaluate and compare self-adaptation techniques and (2) students to study self-adaptation techniques in the realm of a well-understood problem. In addition to formally defining the problem, we have outlined a set of dimensions along which solutions should be evaluated. We have also built ADASIM, an open-source, discrete event simulator for automated traffic routing problem solutions. ADASIM is available for download at http://adasim.googlecode.com and allows for fast and simple implementation and comparison of the solutions. Our vision is that this exemplar becomes part of a benchmark portfolio that helps coordinate and advance self-adaptation research.

REFERENCES

[1] S. M. Blackburn, R. Garner, C. Hoffmann, A. M. Khang, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, A. Phansalkar, D. Stefanović, T. VanDrunen, D. von Dincklage, and B. Wiedermann, "The DaCapo benchmarks:

Java benchmarking development and analysis," in *OOPSLA*, Portland, OR, USA, Oct. 2006, pp. 169–190.

[2] Y. Brun, "Improving impact of self-adaptation and self-management research through evaluation methodology," in *SEAMS*, Cape Town, South Africa, May 2010, pp. 1–9.

[3] A. Carzaniga, A. Gorla, N. Perino, and M. Pezzè, "Automatic workarounds for web applications," in *FSE*, Santa Fe, NM, USA, 2010, pp. 237–246.

[4] B. H. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee *et al.*, "Software engineering for self-adaptive systems: A research roadmap," in *Software Engineering for Self-Adaptive Systems*. Springer-Verlag, 2009, vol. 5525, pp. 1–26.

[5] V. Dallmeier and T. Zimmermann, "Extraction of bug localization benchmarks from history," in *ASE*, Atlanta, GA, USA, 2007, pp. 433–436.

[6] R. de Lemos, H. Giese, H. A. Müller, M. Shaw *et al.*, "Software engineering for self-adaptive systems: A second research roadmap (in press)," in *Software Engineering for Self-Adaptive Systems II*. Springer-Verlag, 2012.

[7] F. Ducatelle, G. A. D. Caro, and L. M. Gambardella, "Cooperative self-organization in a heterogeneous swarm robotic system," in *GECCO*, 2010, pp. 87–94.

[8] A. Elkhodary, N. Esfahani, and S. Malek, "FUSION: A framework for engineering self-tuning self-adaptive software systems," in *FSE*, Santa Fe, NM, USA, 2010, pp. 7–16.

[9] IBM, "Autonomic computing manifesto," http://www.research.ibm.com/autonomic/manifesto, 2001.

[10] T. Kautzmann, M. Wünsche, M. Geimer, H. Schmeck, and S. Mostaghim, "Self-optimizing machine management," in *MCG*, Bonn, Germany, 2010.

[11] "NS-3 network simulator," http://www.nsnam.org.

[12] "OMNeT++ discrete event simulator," http://www.omnetpp.org.

[13] "QualNet network simulator," http://www.scalable-networks.com.

[14] A. J. Ramirez, D. B. Knoester, B. H. Cheng, and P. K. McKinley, "Applying genetic algorithms to decision making in autonomic computing systems," in *ICAC*, Barcelona, Spain, 2009, pp. 97–106.

[15] H. Seebach, F. Nafz, J. Holtmann, J. Meyer, M. Tichy, W. Reif, and W. Schäfer, "Designing self-healing in automotive systems," in *Autonomic and Trusted Computing*, 2010, vol. 6407, pp. 47–61.

[16] "Siemens test suite," http://pleuma.cc.gatech.edu/aristotle/Tools/subjects.

[17] "SIR repository," http://sir.unl.edu.

[18] "SPEC benchmarks," http://www.spec.org/benchmarks.html.

[19] N. M. Villegas, H. A. Müller, G. Tamura, L. Duchien, and R. Casallas, "A framework for evaluating quality-driven self-adaptive software systems," in *SEAMS*, Honolulu, HI, USA, 2011, pp. 80–89.

[20] P. Vromant, D. Weyns, S. Malek, and J. Andersson, "On interacting control loops in self-adaptive systems," in *SEAMS*, Honolulu, HI, USA, 2011, pp. 202–207.

[21] D. Weyns, S. Malek, and J. Andersson, "FORMS: Unifying reference model for formal specification of distributed self-adaptive systems," *TAAS*, 2011.

[22] D. Weyns and M. Georgeff, "Self-adaptation using multiagent systems," *IEEE Software*, vol. 27, pp. 86–91, January 2010.

[23] D. Weyns, R. Haesevoets, A. Helleboogh, T. Holvoet, and W. Joosen, "The MACODO middleware for context-driven dynamic agent organizations," *TAAS*, vol. 5, no. 1, 2010.

[24] D. Weyns, S. Malek, and J. Andersson, "On decentralized self-adaptation: Lessons from the trenches and challenges for the future," in *SEAMS*, Cape Town, South Africa, 2010, pp. 84–93.

[25] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, and J.-M. Bruel, "RELAX: A language to address uncertainty in self-adaptive systems requirement," *Requirements Engineering*, vol. 15, pp. 177–196, 2010.

[26] "ZNN.com exemplar," http://seams.self-adapt.org/wiki/ModelProblem:ZnnExemplar.