CityResolver: A Decision Support System for Conflict Resolution in Smart Cities

Meiyi Ma, John A. Stankovic, Lu Feng
Department of Computer Science,
University of Virginia,
Charlottesville, VA, 22903, USA
Email: {meiyi, stankovic, lu.feng}@virginia.edu

***Abstract*—Resolution of conflicts across services in smart cities is an important yet challenging problem. We present CityResolver – a decision support system for conflict resolution in smart cities. CityResolver uses an Integer Linear Programming based method to generate a small set of resolution options, and a Signal Temporal Logic based verification approach to compute these resolution options' impact on city performance. The trade-offs between resolution options are shown in a dashboard to support decision makers in selecting the best resolution. We demonstrate the effectiveness of CityResolver by comparing the performance with two baselines: a smart city without conflict resolution, and CityGuard which uses a priority rule-based conflict resolution. Experimental results show that CityResolver can reduce the number of requirement violations and improve the city performance significantly.**

***Keywords*-Conflict Resolution, Safety and Performance Requirements, Signal Temporal Logic, Smart City, Smart Services**

I. Introduction

Advances in technologies such as the Internet of Things have transformed the way cities operate. For example, sensors and actuators on streetlights are installed and used to gather information about traffic, street parking and pollution, to adjust LED streetlights to save energy, and to estimate the size of crowds in responding to public disturbances [1]. Various smart services are built to improve the performance and efficiency of smart cities across different domains (e.g., transportation, energy, healthcare). It is estimated that a smart city can generate revenue and cost savings of $2.3 trillion globally through 2024 [2].

In our previous work [3], we identified different types of conflicts among smart services and their severe safety consequences. We also proposed CityGuard [4], a watchdog architecture to detect and resolve conflicts. More recently, we show some preliminary results [5] in the runtime monitoring of smart city safety and performance requirements using Signal Temporal Logic (STL). However, most of our efforts so far have been focusing on conflict detection, while conflict resolution remains an open problem. CityGuard and other solutions [3], [6], [7] adopt a simple approach to conflict resolution by only accepting actions with the highest priority and rejecting others, which does not account for the adaptive policies of action priorities and optimal resolution for city requirements.

Nevertheless, resolution of conflicts across services in smart cities is very complex. The requirements objectives in smart cities are often expressed vaguely in natural language and are potentially conflicting. For example, a resolution that satisfies the requirement of reducing traffic congestion may actually violate another requirement of maintaining noise levels. There is often no best resolution and/or one cannot satisfy all objectives, which implies that trade-offs must occur. In addition, priorities of smart services and actions may be adaptive to the current state of the city. For example, resolving conflicts of traffic congestion may be more urgent at rush hours than at other times. Moreover, there are uncertainties in the state of the city, in human behaviors, and the impact of the resolutions (in time and space). The scale of smart cities and services also makes it challenging to search for an optimal resolution from an enormous solution space.

In this paper, we present CityResolver – a decision support system for conflict resolution in smart cities. CityResolver uses a novel Integer Linear Programming (ILP) based method to generate a small set of candidate resolution options. The ILP-based method makes it more efficient to search for optimal resolutions from an exponentially growing number of possible resolution choices, and it considers adaptive policies of service priorities that change as a function of context. CityResolver then checks these resolution options' impact on city performance using a Signal Temporal Logic (STL) based verification approach. Formalizing city requirements expressed in natural language using STL specification helps to resolve the vagueness of requirement objectives. The verification is performed on predicted traces of future city states, which are generated from the simulation of executing resolution options in the smart city. The simulation also accounts for uncertainties and disturbances in the city (e.g., weather and scheduled events). Given a resolution option, the STL-based approach verifies if the predicted city states violate city requirements. It computes the degree of requirement violations based on three different metrics: (1) the robustness value, (2) the percentage of time when violations occur, and (3) the integral of signal deviations. CityResolver provides a trade-off analysis of different resolution options' effects on various city requirements. The results are plotted on a dashboard

to support decision makers to choose the best resolution. We apply a prototype implementation of CityResolver to a case study of a simulated smart city based on the map of lower Manhattan, New York. We demonstrate the effectiveness of CityResolver by comparing the performance with two baselines: a smart city without conflict resolution, and CityGuard which uses a priority rule based conflict resolution. Experimental results show that CityResolver can reduce the number of requirement violations and improve the city performance significantly.

Contributions. The major contributions of this paper are:

1) Design, implementation and evaluation of CityResolver – a decision support system for conflict resolution in smart cities.
2) An Integer Linear Programming based method to generate resolution options.
3) A Signal Temporal Logic based verification approach to compute smart city requirement violation degrees using three different metrics.

Paper Organization. The rest of the paper is organized as follows. We introduce an motivating example of a smart city and its services in Section II, and describe an overview of CityResolver in Section III. Then, we present the ILP-based method for generating candidate resolution options in Section IV and a STL-based approach for computing option trade-offs in Section V. We demonstrate the usefulness of our proposed system via experimental evaluations in Section VI. We survey the related work in Section VII and draw concluding remarks in Section VIII.

II. Motivating Example

In this section, we describe an example of smart services and their conflicts, based on a map of the lower Manhattan district in the New York City (Figure 1). We assume that there are 10 different smart services in this district, which are overseen by a smart city operations center. We only introduce four services here, and list all ten services in Table I (see Section VI). **S1** is a smart traffic service, which can control traffic signals in street intersections to relieve congestion and optimize or improve traffic performance. **S2** is a smart emergency service, which can request green traffic signals in order to transport patients in critical conditions to hospitals as soon as possible. **S3** is a smart accident service, which can block a street where some accident occurs and alert nearby vehicles to detour. **S4** is a smart infrastructure service, which can schedule infrastructure check-up and repair appointments. The operations of these smart services have to satisfy a set of safety and performance requirements in the smart city. For example, **R1** is an environment requirement that the noise level in the school area should always be less than 50db. **R2** requires that the Carbon Monoxide (CO) emission in an intersection should always be less than 40mg. **R3** is a requirement for transportation

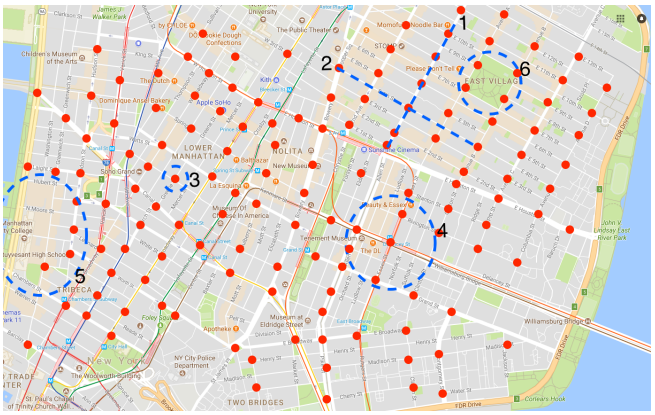


Figure 1. A map of the lower Manhattan district in the New York City. Red dots: street intersections with hypothetical smart sensors and services. Blue dashed lines/circles: areas of interest (1: 1st Avenue, 2: East 4th Street, 3: The intersection of Greene Street and Grand Street, 4: Blocks around the entry of Williamsburg Bridge, 5: Stuyvesant High School and Borough of Manhattan Community College, 6: East Village).

domain, which requires that the waiting time of the traffic in an intersection should not be greater than certain threshold λ. **R4** specifies that an emergency vehicle should not wait in an intersection for more than 10 seconds. A complete list of requirements considered in this paper can be found in Table II in Section VI.

Conflicts may arise when different smart services request contradicting actions on the same actuators at the same time, or when the effects of service actions violate the smart city's safety and performance requirements. For example, suppose that S1 requests longer green traffic signals on 1st Avenue to relieve traffic congestion, while S2 requests green traffic signals on East 4th Street to transport a patient in an emergency. The requested actions by S1 and S2 contradict each other on the traffic signal at the interaction of 1st Avenue and East 4th Street (annotated as areas 1 and 2 in Figure 1). In addition, the effects of actions requested by S1 and S2 may cause an increased traffic in nearby East Village (area 6 in Figure 1), and thus violating requirements about noise levels (R1) and CO emission (R2).

The smart city operations center detects such conflicts and provides resolutions by accepting or rejecting smart services' action requests. However, it is very challenging to find an optimal or even acceptable conflict resolution. First, the feasible set of resolutions grows exponentially with the increasing number of smart services and actions. An exhaustive search over the entire solution space is not efficient or even possible. Second, what does it mean by optimal when considering multiple smart city requirements that are expressed vaguely in English? For example, one resolution may dramatically reduce traffic congestion, but increase pollution levels. An ideal resolution should balance the trade-off between multiple objectives. Third, the severity of conflicts and the importance of service actions are often

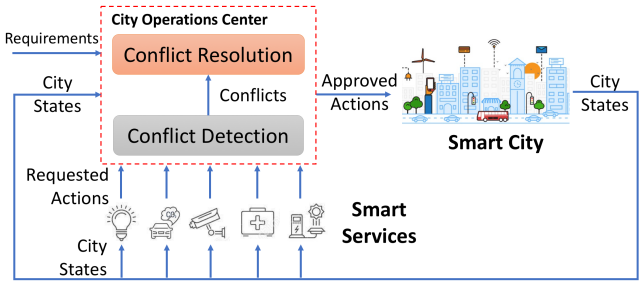


Figure 2. Overview of conflict detection and resolution among smart services in smart cities.

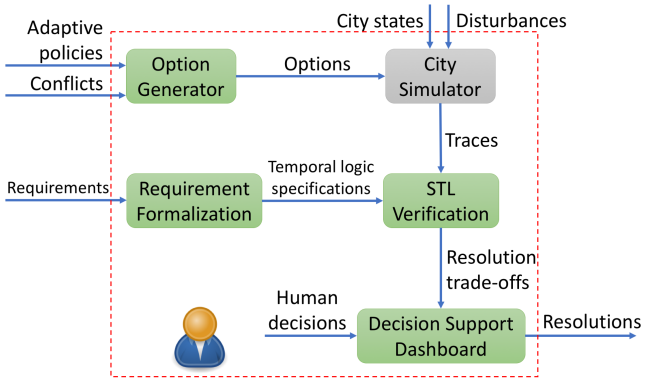


Figure 3. Overview of CityResolver – a decision support system for conflict resolution in smart cities.

dependent on the current state of the city. For example, during rush hour, resolving traffic congestion is more urgent than maintaining noise levels and CO emissions. Thus, the conflict resolution should consider an adaptive policy of prioritizing smart services and requirements. Finally, there are many uncertainties in the state of the city, including disturbances that are predictable (e.g., weather, events) and unpredictable events (e.g., accidents). To address these challenges, we propose a decision support system for conflict resolution in smart cities as described in the next section.

III. System Overview

We envision a watchdog architecture (e.g., CityGuard [4]) in which a city operations center would oversee all smart services, detect conflicts among service requests, and provide resolutions. Such a city operations center could follow real-world prototypes including IBM's Rio de Janeiro Operations Center [8] and Cisco's Smart+Connected Operations Center [9], where real-time information about city states (e.g., traffic, pollution) are collected from citywide sensors and displayed on the command room's monitors. Figure 2 shows an overview of our envisioned architecture that extends the functionality of a city operations center with conflict detection and resolution. Smart services send action requests based on real-time city states. The city operations center intercepts these action requests and detects if there is any conflict that would lead to contradicting actions or violations of city requirements. If no conflict is detected, all action requests are approved for execution in the smart city. Otherwise, an optimal resolution is computed to resolve the detected conflicts. We refer readers to our previous papers [4], [5] for conflict detection methods, and focus on addressing challenges of conflict resolution.

We present CityResolver – a decision support system for conflict resolution in smart cities. An overview of CityResolver is shown in Figure 3. Suppose one or more conflicts between smart services' action requests are detected. The first step is to generate a set of resolution options, each of which may accept a subset of action requests and reject the others. The resolution options may also suggest alternatives or delayed executions of requested actions. Thus, the number of potential resolution options grows exponentially with the number of action requests. We develop an Integer Linear Programming (ILP) based method to select a small set of candidate options, accounting for policies that define the priorities of the smart services and their actions. These policies are not fixed, but are adaptive based on current city states (context). We will describe the ILP-based method in Section IV.

The second step is to simulate the execution of these resolution options in order to predict the effect of choices on the city. Here, we use an off-the-shelf city simulator [10]. Multiple simulations may be instantiated in parallel to simulate the execution of smart city under different resolution options. For each resolution option, the simulator starts with the city states at the current time t (i.e., when conflicts are detected) and simulates the city executing the resolution option for a period of Δ_t into the future. Traces (time series) of city states from t to $t + \Delta_t$ are generated for verification that an option does not violate safety and performance requirements. The simulation also accounts for disturbances in the city (e.g., heavier traffic during 5 to 7 pm, a 80% chance of rainy day, or a big event is scheduled). We distinguish two types of disturbances or uncertainties in smart cities: predictable and unpredictable. The simulation only considers predictable disturbances. However, unpredictable disturbances (e.g., accidents, device failures) are handled in CityResolver due to a continuous feedback loop that is monitoring city states in real-time (see Figure 2). If these states change greater than associated set points then the services themselves issue new actions which we intercept and re-apply the detection and resolution actions.

The next step is to verify if the simulated traces of city states of each option satisfy various city requirements. We develop an approach to formalize smart city requirements as Signal Temporal Logic (STL) specifications and compute the trade-off between different resolution options on multiple specification objectives via STL verification (see Section V).

The trade-offs between options are displayed in a decision

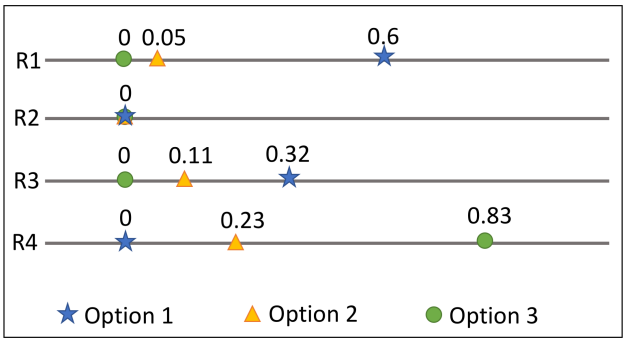


Figure 4. An example dashboard displaying the trade-off between three resolution options in terms of the percentage of time violating R1-R4. (Example 1 describes these options.)

support dashboard. The decision maker chooses a resolution, by comparing the performance of different options on various city requirements. For example, Figure 4 shows the trade-off between three options on requirements R1-R4. We will describe these options and their generation later in Example 1 in Section IV. The values in the trade-off display represent the violation degree in terms of the percentage of time when the requirement is violated. The zero value means that the requirement is not violated. Figure 4 shows that Option 3 satisfies the first three requirements but violates R4 most of the time, while Option 1 satisfies R2 and R4, but violates R1 and R3. The human decision maker may choose a resolution option based on individual preferences. To reduce the human burden of selecting resolutions every time a conflict occurs, CityResolver also allows the automatic selection of optimal resolutions based on a set of rules predefined by the human decision makers. Suppose that a human decision maker thinks that R4 is more important than other requirements and defines a rule that the optimal resolution should not violate R4. Then Option 1 is automatically selected based on the rule.

IV. Generating Resolution Options

In this section, we present an Integer Linear Programming (ILP) based method to generate a small set of resolution options, which corresponds to the "Option Generator" module in Figure 3.

Suppose that there are m on-going smart service actions executing in the smart city without any conflict. The city operations center intercepts n new action requests from smart services and detects that there are some conflict between these $m+n$ actions. One strategy to obtain resolutions is to only accept some of the new actions while rejecting others in a way that there are no conflicts. To achieve this it may also be necessary to suspend some on-going actions. Thus, the number of possible resolution choices is at least 2^{m+n}. If we consider a more complex resolution strategy, such as suggesting alternative actions to the requested actions or putting the rejected actions in a waiting list for delayed execution, then the solution space of possible resolutions becomes even larger. It would be very challenging, if not impossible, to check all these resolution choices' impact on city states and requirements within the short time frame of resolution decision making. Thus, we present a method to select a small number of candidate resolution options based on the intuition that a good resolution should (1) accept as many actions as possible, (2) not allow contradicting actions, and (3) account for priorities of services and actions.

We formulate the problem as an integer linear program. Given a set A of smart service actions causing conflicts, we define a binary variable $\mu \in \{0,1\}$ for each action $a \in A$ to track if the action is chosen by a candidate resolution option. Each action a is associated with a weight value $w \in \mathbb{Z}$, representing the action priority determined by current, state-dependent importance policies. For simplicity, we assume that action weights are given as constants at time t. We denote a set C of contradicting action pairs and a set D of dependent action pairs. We also group an action and its alternatives into a set $\theta \subseteq A$. The resulting ILP problem is

$$\underset{w_i \in \mathbb{Z},\ \mu_i \in \{0,1\}}{\text{maximize}} \sum_{1 \le i \le |A|} w_i \times \mu_i \tag{1a}$$

subject to

$$\forall (a_i, a_j) \in C: \ \mu_i + \mu_j \le 1, \tag{1b}$$

$$\forall (a_i, a_j) \in D: \ \mu_i - \mu_j = 0, \tag{1c}$$

$$\forall a_i \in \theta \subseteq A: \ \sum \mu_i \le 1 \tag{1d}$$

The objective function (1a) is to maximize the number of accepting actions in the resolution based on their priority weights. The constraint (1b) guarantees a resolution does not accept a pair of contracting actions. The constraint (1c) ensures that dependent actions are both accepted or rejected at the same time. Finally, the constraint (1d) requires that at most one action from a set of alternative actions is chosen by a resolution. Transforming the problem to ILP and solving it with the Gurobi tool do not necessarily find the best solution when the number is very large, but it can give the solution in polynomial time, which is very important for runtime decision making system in cites.

We illustrate the usage of the ILP solution below.

Example 1: Suppose that smart traffic service S1 requests seven traffic signals on the 1st Avenue to stay green for 5 minutes. The requested actions are denoted as $\{a_1 ... a_7\}$, corresponding to traffic signals drawn as seven red dots (from south to north) on street 1 in Figure 1. If action a_3 is not accepted, the service also allows an alternative action (denoted by a_8) to keep the corresponding signal green only for 3 minutes. Suppose that, at the same time, the smart emergency service S2 requests three green traffic signals on the East 4th Street for 3 minutes. The actions are denoted as $\{a_9, a_{10}, a_{11}\}$, corresponding to traffic signals

drawn as three red dots (from west to east) in street 2 in Figure 1. Actions a_9 and a_{10} are interdependent. Action a_{10} is contradicting with actions a_3 and a_8. Actions requested by the emergency service S2 has a higher priority weight than the traffic service S1. Let the weight value for S2 actions be 2 and the weight value for S1 actions be 1. We write an ILP as follows:

$$\underset{\mu_i \in \{0,1\}}{\text{maximize}} \sum_{1 \le i \le 8} \mu_i + \sum_{9 \le i \le 11} 2 \times \mu_i$$

subject to

$$\mu_3 + \mu_{10} \le 1$$
$$\mu_8 + \mu_{10} \le 1$$
$$\mu_9 - \mu_{10} = 0$$
$$\mu_3 + \mu_8 \le 1$$

We rank solution results based on their objective function values. The top 3 resolution options are as follows.

- Option 1: Reject a_3 and a_8, accept other actions.
- Option 2: Reject a_3, a_8 and a_{11}, accept other actions.
- Option 3: Reject a_8, a_9 and a_{10}, accept other actions.

A trade-off between these options is shown in Figure 4.

V. Verifying Resolution Options

In this section, we describe how to compute the trade-off between different resolution options via Signal Temporal Logic (STL) based runtime verification of city requirements. We introduce STL and the formalization of smart city requirements in Section V-A, and present methods for computing requirement violation degrees in Section V-B.

A. Requirement Formalization

Signal Temporal Logic (STL) [11] is a formalism used to specify real-time properties of discrete and continuous signals. The syntax of an STL formula φ is usually defined as follows,

$$\varphi ::= \mu \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Diamond_{(a,b)}\varphi \mid \Box_{(a,b)}\varphi \mid \varphi \mathbf{U}_{(a,b)}\varphi.$$

We call μ a signal predicate, which is a formula in the form of $f(x) > 0$ with a signal variable $x \in \mathcal{X}$ and a function $f : \mathcal{X} \to \mathbb{R}$. The temporal operators $\Box$, $\Diamond$, and $\mathbf{U}$ denote "always", "eventually" and "until", respectively. The bounded interval (a, b) denotes the time interval of temporal operators and can be omitted if the internal is $[0, +\infty)$. Formula $\Box_{(a,b)}\varphi$ is true iff φ is always true in the time interval (a, b). Formula $\Diamond_{(a,b)}\varphi$ is true iff φ is true at sometime between a and b. Formula $\varphi_1 \mathbf{U}_{(a,b)}\varphi_2$ is true iff φ_1 is true until φ_2 becomes true at sometime between a and b. We refer readers to [11] for the formal definition of STL semantics.

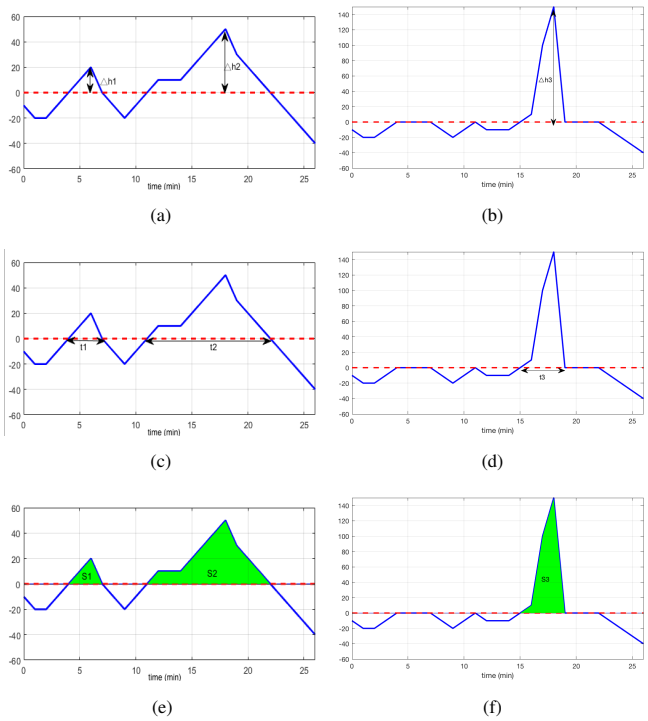


Figure 5. An example of two options showing three metrics of violation degrees for STL formula $\Box_{(0,25)}$ (Noise < 50). Blue solid curves represent the signal (Noise $- 50$) under resolution option (i) and (ii). For option (i), (a) the robustness value: Δh_2, (c) the percentage of violation time: $(t_1 + t_2)/25$, and (e) the integral of deviation: $S_1 + S_2$; For option (ii), (b) the robustness value: Δh_3, (d) the percentage of violation time: $t_3/25$, and (f) the integral of deviation: S_3.

In our previous work [5], we provide a set of templates for expressing typically safety and performance requirements in smart cities as STL specifications. We find that most smart city requirements can be specified using STL formula in the form of $\Box_{(a,b)}$ $(x < \lambda)$ where x is a signal about city state and λ is a threshold.

Example 2: We translate requirements R1-R4 (described in Section II) to STL formulas as follows.

- STL formula for R1: $\Box_{(0,\Delta t)}$ (Noise < 50).
- STL formula for R2: $\Box_{(0,\Delta t)}$ (CO < 40).
- STL formula for R3: $\Box_{(0,\Delta t)}$ (WaitTime $< \lambda$).
- STL formula for R4: $\Box_{(0,\Delta t)}$ (EmergencyTime < 10).

B. Computing Requirement Violation Degrees

We now describe how to compute the degree in which a continuous signal about smart city state violates a city requirement $\Box_{(a,b)}$ $(x < \lambda)$. A notion of robustness value for satisfying or violating STL formulas is formally defined in [11]. The robustness value of a given signal x violating $\Box_{(a,b)}$ $(x < \lambda)$ at time τ is defined as

$$\rho = \sup_{t \in (\tau+a, \tau+b)} (x(t) - \lambda). \quad (2)$$

Intuitively, the robustness value indicates extremum points of the signal. The robustness value is useful for telling the

worst-case performance, but it does not show the average or overall performance. For smart cities requirements, we are interested to know both. We use the following example to illustrate why measuring the robustness value only is not sufficient for finding optimal resolutions.

Example 3: Suppose that the noise level is between 50 db and 100 db for 12 min with option (i), and over 100 db for 2 min with option (ii). We verify the city state against requirement $\square_{(0,25)}$ $(\mathsf{Noise} < 50)$. Figure 5(a) plots the signal of $(\mathsf{Noise} - 50)$ under option (i) and indicates that the robustness value of violating the requirement is Δh_2 – the maximum deviation from the threshold within the time interval. Figure 5(b) plots the signal of $(\mathsf{Noise} - 50)$ under option (ii) where the robustness value of violating the requirement is Δh_3. Option (i) has a better performance than option (ii) in terms of the robustness value of violating the requirement, because $\Delta h_2 = 50$ and $\Delta h_3 = 150$. However, a decision maker may actually find option (ii) a better resolution, because the noise level only exceeds the threshold 50 db for a shorter period of time (2 min instead of 12 min). Thus, smaller value of robustness violation degree sometimes does not imply a better resolution.

To address this limitation, we present two new metrics for measuring the degree of violating smart city requirements specified in STL: (1) the percentage of time when a requirement is violated, and (2) the integral of signal deviations. To start with, we define Equations 3 and 4 to calculate the positive part (denoted by $\theta^+(x)$) and the negative part (denoted by $\theta^-(x)$) of a function $f(x)$, respectively.

$$\theta^+(x) = \max(f(x), 0) = \begin{cases} f(x) & \text{if } f(x) > 0 \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

$$\theta^-(x) = \min(f(x), 0) = \begin{cases} f(x) & \text{if } f(x) < 0 \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

We compute the percentage of time when a given signal x violating $\square_{(a,b)}$ $(x < \lambda)$ as follows:

$$\eta = \frac{1}{b-a} \int_{\tau+a}^{\tau+b} \text{sgn}(|\theta^-(x(t) - \lambda)|)dt. \quad (5)$$

We use Equation 6 to compute the integral of signal deviations accumulated in a period when the requirement $\square_{(a,b)}$ $(x < \lambda)$ is violated.

$$\gamma = \int_{\tau+a}^{\tau+b} (|\theta^-(x(t) - \lambda)|)dt. \quad (6)$$

Example 4: Figure 5(c) and Figure 5(d) shows the percentage of time when requirement $\square_{(0,25)}$ $(\mathsf{Noise} < 50)$ is violated under option (i) and (ii), respectively. Option (ii) has a better performance with this metric of requirement violation degree, because $\frac{(t_1+t_2)}{25} < \frac{t_3}{25}$.

Figure 5(e) and Figure 5(f) shows the integral of noise level deviations when requirement $\square_{(0,25)}$ $(\mathsf{Noise} < 50)$ is violated under option (i) and (ii), respectively. The integral value for option (i) is the sum of areas $S_1 + S_2$, while the integral value for option (ii) is S_3. It turns out that $S_1 + S_2 = S_3 = 26$. Thus, option (i) and option (ii) have the same performance with this metric of requirement violation degree.

These three metrics are useful to evaluate different smart city requirements. Robustness value is more suitable for requirements with hard constraints, such as the emergency vehicle waiting time and the number of accidents. Most city requirements are soft constraints, which do not require the signal strictly within a threshold bound. In such cases, the percentage of violation time is an important measurement, especially for environmental signal like the pollution level. The integral metric combines robustness and violation time. Therefore, it helps to compare the overall performance between different signals. For example, calculating the integral of violating requirements in the transportation domain (e.g., the waiting number and waiting time of vehicles) helps to reveal the congestion degree.

VI. Experimental Evaluation

A. Experiment Setup

We implement CityResolver and apply it to a case study of a simulated smart city based on the lower part of Manhattan in New York. The transportation state is generated from the Traffic volume counts of New York city data [12]. This data set contains the traffic volumes from 160 streets in Manhattan during 2013-2014. Analyzing the data set, the average traffic volume on all streets is 105,397 vehicles and 658 vehicles per street per hour, making the in-coming vehicle rate at 5.5 second per vehicle.

We implement a smart city simulator using the data set and the Simulation of Urban MObility (SUMO) [10], a transportation simulator which allows modelling of traffic systems including road vehicles, public transport and pedestrians. We build ten smart services (see Table I) running over 140 locations of lower Manhattan, New York, as shown in Figure 1. We verify resolution options with the same city safety and performance requirements in detection component, which are listed in Table II. Different types of requirements are suitable to different space ranges and distributed over even more than 140 locations in the map. We use CityGuard [4] to detect conflicts and CityResolver to provide resolutions for those conflicts. To generate resolution options, CityResolver uses the Gurobi optimization tool [13] to solve integer linear programs.

The experiments are evaluated on a server machine with 16 core CPUs; each core is 3.1GHz. The operating system is Ubuntu 14.04.5.

Table I
LIST OF SERVICES RUNNING IN SIMULATED MANHATTAN

Service	Description
S1: Traffic Service	It controls traffic signals in street intersections to relieve congestion and optimize or improve traffic performance.
S2: Emergency Service	It requests green traffic signals in order to transport patients in critical conditions to hospitals as soon as possible.
S3: Accident Service	It blocks a street where some accident occurs and alert nearby vehicles to detour.
S4: Infrastructure Service	It schedules infrastructure check-up and repair appointments.
S5: Pedestrian Service	It shortens the pedestrians' waiting time by adjusting traffic signals when pedestrians wait in the intersection.
S6: Air Pollution Control	It adjusts the traffic by adjusting traffic signal and sending speed request to vehicles when CO emission is high.
S7: PM2.5/ PM10 Control	It adjusts the traffic when PM2.5/ PM10 emission is high by adjusting traffic signal and sending speed request to vehicles directly.
S8: Parking Service	It directs the driver to the nearest parking lot.
S9: Noise Control	When noise level exceeds its threshold, it controls the number of vehicles going through related streets and redirect vehicles on the streets by adjusting traffic signals.
S10: Event Service	It ensures operation of a city event by blocking the lanes nearby the event.

Table II
LIST OF REQUIREMENTS (THE CHECK MARK INDICATES THE SUITABLE SPACE RANGE OF THE REQUIREMENT IN PRACTICE.)

Requirement	Intersection	Street	Block
R1: The noise level in the school area should always be less than 50db.	✓		✓
R2: The CO emission in an intersection should always be less than 40mg.	✓		✓
R3: The waiting time of the traffic in an intersection should not be greater than certain threshold λ.	✓	✓	
R4: An emergency vehicle should wait in an intersection for more than 10 seconds.	✓		
R5: No vehicle collision should occur.	✓	✓	
R6: The number of vehicles in a street should never exceed its maximum vehicle capacity.		✓	
R7: The traffic yield number in a street should not increase by certain threshold λ.		✓	
R8: The number of pedestrians waiting in an intersection should not be greater than certain threshold λ.	✓		
R9: Emergency vehicles should not be directed to a blocked lane or area.		✓	
R10: The noise level in a street should always be less than 70 dB.		✓	✓
R11: The hydrocarbons (HC) emission in a lane should always be no more than 1 mg.		✓	✓
R12: The particulate matter (PMx) emission in a lane should always be no more than 0.2 mg.		✓	✓

B. Results on Option Generator

The total number of resolution options increases exponentially with the number of actions. That is, given n actions, there are 2^n possible choices of resolutions. CityResolver builds an ILP-based option generating model, which is able to identify a set of optimal options regarding the sum of action requests multiplied by their weights. We tested the option generator for over 500 conflicts and where each conflict has up to 100 actions. The computation time of generating resolution options for each conflict is within 1 second using CityResolver.

C. Results on Computing Violation Degree

CityResolver provides the three metrics (see Section V.B) for calculating the violation degree. For example, in this study we measure the violation degree on 4 requirements (R1 - R4) using the three metrics. The traces of each option are generated from the city simulator. When predicting the future city states, we incorporate the known disturbance factors. For example, one of our results shown in Figure 6 considers that rush hour is coming, which will cause heavy traffic. Therefore, in our simulation, we increase the traffic volume at that future time to obtain a more accurate prediction. We observe the following based on Figure 6.

- There are trade-offs between different options using all three metrics. For example, in (a), Option 4 has the best performance on R3 (i.e., the congestion requirement), but has the worst performance on R2 (i.e., the noise requirement). The other three options, though not best on all requirements, have a more even distribution.
- Different metrics lead to different results: Option 4 is the best in (a) and (b) regarding requirement R3, but is the second best in (c).
- For the robustness metric (a) and integral metric (c), it is difficult to compare the performance of the same option on different requirements, because measurement scales are not necessarily the same. However, it is uniform using the percentage of time (b), where all violation degrees are on scale of 0 to 1.
- All three metrics can help a decision maker to compare the performance of different options on the same requirement. They may choose the metric based on the requirement type and the context.
- It is helpful to compare more than one metric, because they represent different properties of the requirement. For example, Option 4 has a worst performance on R2 measured by the robustness and integral metrics, but does not have significant violation on the percentage of time metric. The results also indicate that Option 4 exceeds the noise level threshold for only a few times within this period, but each time the difference is large.

D. Overall Performance

We evaluate the city performance with CityResolver across domains of transportation, environment and emergency. Specifically, the performance evaluation uses metrics including the number of violated requirements per conflict,

Table III
COMPARISON ON THE CITY PERFORMANCE WITH CITYGUARD AND WITH CITYRESOLVER

Case	System	Number of Violated Requirements	CO (mg)	Noise (db)	Emergency Waiting Time (s)	Vehicle Waiting Number	Pedestrian Waiting Time (s)
Case 1: [S1, S2]	None	20	53.12	67.73	21.00	23	65.00
	CityGuard	7	67.90	72.42	**6.70**	32	60.00
	CityResolver	**0**	**39.70**	**46.54**	9.80	**22**	**35.90**
Case 2: [S1, S2, S3]	None	28	55.20	49.00	31.20	19	83.00
	CityGuard	18	54.20	62.00	11.00	21	88.00
	CityResolver	**2**	**44.30**	**48.90**	**8.70**	**15**	**63.50**
Case 3: [S1, S8, S10]	None	16	53.91	67.00	9.20	23	53.20
	CityGuard	**0**	49.20	48.70	8.80	24	50.10
	CityResolver	**0**	**30.80**	**40.80**	**7.80**	**18**	**49.20**
Case 4: [S6, S7, S9, S10]	None	15	48.30	59.00	14.50	42	79.20
	CityGuard	8	**38.60**	62.10	13.80	34	76.30
	CityResolver	**1**	39.50	**56.30**	**8.30**	**29**	**65.20**
Case 5: [S2, S3, S4, S5, S9]	None	39	87.30	45.00	7.40	39	68.30
	CityGuard	32	90.30	46.00	7.80	39	65.70
	CityResolver	**4**	**62.70**	**43.00**	**6.70**	**28**	**59.60**

emission of CO, noise, the waiting time for the emergency vehicle, the waiting number of vehicles, and the waiting time for pedestrians. The number of violated requirements indicates that when detecting one conflict or verifying a solution, how many requirements are violated by this conflict or option. The remaining metrics are calculated by the average value at one intersection. For example, if the effective range of one conflict is 3 blocks (i.e. involving 9 intersections), then the value is the average value of these 9 intersections.

We compare the result with two baselines: (1) a smart city without any controller for conflict resolution, and (2) CityGuard. To the best of our knowledge, there is no other existing solution for conflict resolution in smart cities. CityGuard focuses on the conflict detection and implemented a simple priority-based conflict resolver, i.e. when there is a potential conflict between different action requests, it only accepts the action with the highest priority. Also, it does not verify the performance of selected actions. The experiments are conducted on the lower Manhattan with 10 smart services running over 140 locations. The results from 5 cases are shown in Table III, from which we have the following observations.

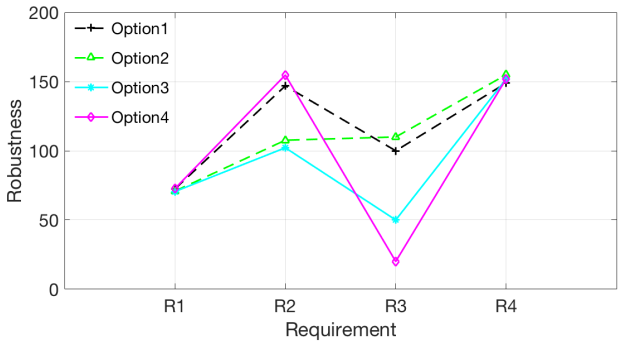
Case 1: Conflict happens between the S1 smart traffic service and the S2 smart emergency service. Without resolution, it violates 20 city requirements. With CityGuard, it detects the conflict and accepts the actions of the emergency service, which reduce the number of violated requirements and improve city performance regarding the emergency vehicle waiting time. However, it harms the performance on CO and noise with a bias in favor of the emergency. Instead, CityResolver finds an optimal solution, which reduces the number of violated requirements to 0. Compared with CityGuard, CityResolver also improves other metrics, for example, it reduces the CO emission by 41.5%.

Case 2: Another conflict happens among S1 (traffic service), S2 (emergency service) and S3 (accident service). Similar to Case 1, both CityGuard and CityResolver propose a resolution and improve the city performance. Two things to be noted, 1) CityResolver has a better overall performance than CityGuard, 2) CityResolver also does not find a resolution satisfying all requirements, but it reduces the violation number by 13 times and 8 times compared to the none-control system and CityGuard, respectively.

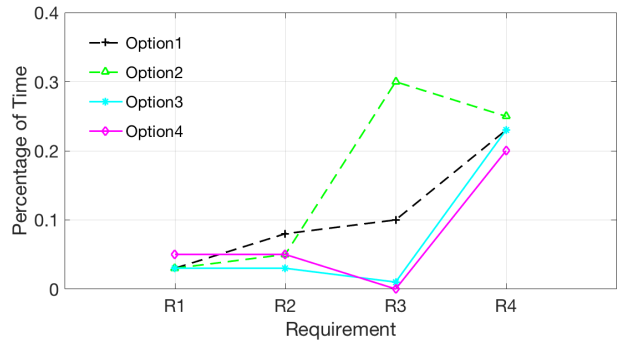
Case 3: It demonstrates a conflict among traffic, parking and event services, where both CityGuard and CityResolver find a resolution that satisfies all requirements. However, CityResolver still has better performance, for example, it beats CityGuard by 37.4% on CO and 25% on the vehicle waiting number. The reason is that CityResolver accepts more actions if possible, which as a result, benefits the city.

Case 4 and *Case 5* evaluate cases under a larger number of services and actions. CityResolver finds potential options and verifies them within a reasonable time. In particular, it maintains a small number of violated requirements comparing to baselines. By showing trade-offs between the requirements that have to be violated, CityResolver gives the city manager a chance to select a better choice based on the context. Meanwhile, though not largely, CityResolver improves performance regarding each domain by 6.5% to 30.6% comparing to CityGuard.

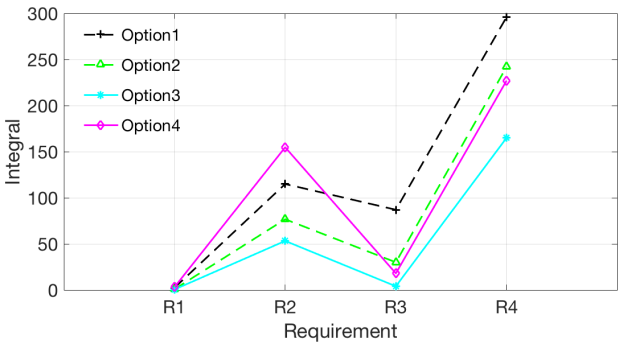
In summary, CityResolver reduces the number of violations significantly and improves the overall city performance without sacrificing the performance on other metrics. However, to be noted, CityResolver does not always have the best performance over CityGuard. For example, in Case 1 and Case 4, its performance on the emergency vehicle waiting time and CO emission are worse than CityGuard. The reason is that the emergency service and air quality service have a higher priority and so do their actions with CityGuard resolution. With this bias, it does not have good performances on other metrics. On the contrary, CityResolver values high-weight actions as well as optimizes other metrics.



(a) Metric: Robustness violation value



(b) Metric: Percentage of violating time



(c) Metric: Integral of signal deviations

Figure 6. Trade-offs of four options based on their performance on requirements R1-R4

VII. Related Work

A survey paper on the conflict detection and resolution in smart home and building automation [14] categorizes conflict resolution into four categories by its solvability from high to low as conflict avoidance, conflict resolution, acknowledge inability of resolving and acknowledge occurrence. Our approach focuses on the conflict avoidance and resolution acknowledgement by predicting and detecting conflicts, and showing the trade-offs among potential resolutions. With effective cooperation, conflicts are prevented with the best option.

The general problem of conflict detection and resolution has been studied for applications such as smart homes, buildings and multi-agent systems, in addition to smart cities. DepSys [6] and SIFT [7] provide comprehensive strategies to specify, detect, and resolve conflicts in a smart home setting. To resolve the conflict, they categorize smart apps into different priorities based on their domains, for example, health apps have higher priority than energy apps. Similarly, our previous work [3], [4] categorizes various safety & performance requirements in smart cities, and proposes the CityGuard architecture for conflict detection and resolution. Focusing on the detection, CityGuard uses a simple priority-based resolution, which improves the performance in domains of transportation and environment. However, in reality, the priorities of services are adaptive to the context. A service important to one context is not necessarily important to all the cases. In addition, action requests can be independent from each other, so it is not efficient to reject all actions from a lower priority service under all circumstances. Comparing with above works, this paper generates the resolution options based on the relationship between actions and their adaptive policies, which considers the importance of an action with the real context and optimizes accepted options.

Researchers also propose different ways for conflict resolution based on its conflict types [14]. For example, for interest conflict, [15] considers the application's demands for quality of services and resource consumption, and uses a client-server architecture model to select the conflict resolution, which, however, can only deal with very small scale clients. [16] builds a resource management method to resolve the conflict in smart buildings. [17] builds an ontology-based policy framework using an AI planner to detect and automatically resolve policy conflict. However, these methods focus on detecting and resolving direct conflicts. They do not (1) consider the secondary effects on the environment, which are where most of city conflicts arise, or (2) predict and avoid conflicts. The environment of smart cities is so complicated that the conflicts among smart services not only include all the above conflict types, but also contain different types of environmental conflicts. Rule-based or resource management approaches are not sophisticated enough to predict and prevent conflicts in smart cities. Our approach uses a feedback control loop which monitors city states in real time, and predicts the effects of actions and the potential resolution options by predicting and verifying future city traces.

VIII. Concluding Remarks

In this paper, we build CityResolver – a decision support system for conflict resolution in smart cities. Using an Integer Linear Programming based method, CityResolver first generates a small set of potential resolution options considering the dependency of actions and adaptive policies of their services. It verifies the set of options with an

STL based approach and calculates the effects of options on different requirements taking the disturbance factors into account. Then it shows the trade-offs between resolution options in a dashboard supporting decision makers to choose the best resolution. The evaluation results show that, CityResolver is able to reduce the number of violation requirements significantly comparing to the smart city without a controller and with a priority based resolver. For example, CityResolver reduces the number from 39 down to 4, while CityGuard reduces the violation number from 39 to 32. Moreover, CityResolver improves city performance comparing to the baselines. For example, in our experiments, it beats CityGuard on the CO emission up to 37.4% and the congestion by 25%.

The future work includes the following directions:

Smart City Requirements. There are currently limited number of smart city requirements available, most of which are in the form of "(city state) always less than (threshould)". For example, "the noise level should always be less than 50 db", "the emergency vihicle waiting time should always be less than 10 second". Thus, we only considered the computation of violation degrees for this type of requirements. We plan to collaborate with smart cities to explore more variety of city requirements. We are also developing tools for the automated translation from natural language requirements to temporal logic based formal specifications.

Option Generator. The ILP-based option generator uses an objective function that considers the dependency between actions and the adaptive weights of services. However, the optimal set obtained from the generator does not necessarily contain the corner case option, i.e. the option that has the best performance on one or more requirements. The corner case options can help decision maker to compare trade-offs when selecting a resolution option. We will explore heuristics to generate options that cover the best performance on different requirements.

Disturbance Model. CityResolver accounts for known disturbance factors (e.g., rainy weather, the rush hour, the big event) when predicting future city states. We will develop more comprehensive disturbance models that take into account the probability of a disturbance event and quantify their effects on city conflict detection and resolution.

ACKNOWLEDGMENT

This work was funded, in part, by NSF under grants CNS-1527563 and CNS-1319302.

REFERENCES

[1] WSJ, "The rise of the smart city," 2017. [Online]. Available: https://www.wsj.com/articles/the-rise-of-the-smart-city-1492395120

[2] Cisco, "The digital value of smart cities," 2017. [Online]. Available: https://discover.cisco.com/en/us/iot/whitepaper/smart-cities-digital-value

[3] M. Ma, S. M. Preum, W. Tarneberg, M. Ahmed, M. Ruiters, and J. Stankovic, "Detection of runtime conflicts among services in smart cities," in *Proceedings of IEEE International Conference on Smart Computing*. IEEE, 2016, pp. 1–10.

[4] M. Ma, S. M. Preum, and J. A. Stankovic, "Cityguard: A watchdog for safety-aware conflict detection in smart cities," in *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, 2017, pp. 259–270.

[5] M. Ma, J. A. Stankovic, and L. Feng, "Runtime monitoring of safety and performance requirements in smart cities," in *1st ACM Workshop on the Internet of Safe Things*, 2017.

[6] S. Munir and J. Stankovic, "Depsys: Dependency aware integration of cyber-physical systems for smart homes," in *Proceedings of ACM/IEEE International Conference on Cyber-Physical Systems*, 2014.

[7] C.-J. M. Liang, B. F. Karlsson, N. D. Lane, F. Zhao, J. Zhang, Z. Pan, Z. Li, and Y. Yu, "Sift: building an internet of safe things," in *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*, 2015, pp. 298–309.

[8] N. Y. Times, "Ibm takes 'smarter cities' concept to rio de janeiro," 2012. [Online]. Available: http://www.nytimes.com/2012/03/04/business/ibm-takes-smarter-cities-concept-to-rio-de-janeiro.html

[9] Cisco, "Smart+connected operations center." [Online]. Available: https://www.cisco.com/c/en/us/solutions/industries/smart-connected-communities/city-operations-center.html

[10] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "Sumo–simulation of urban mobility: an overview," in *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind, 2011.

[11] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *Proceedings of International Conference on Formal Modeling and Analysis of Timed Systems*, 2010, pp. 92–106.

[12] *New York City Open Data*, https://nycopendata.socrata.com/.

[13] "Gurobi Optimization," http://www.gurobi.com.

[14] S. Resendes, P. Carreira, and A. C. Santos, "Conflict detection and resolution in home and building automation systems: a literature review," *Journal of Ambient Intelligence and Humanized Computing*, vol. 5, no. 5, pp. 699–715, 2014.

[15] R. B. S. Thais, B. R. Linnyer, and A. L. Antonio, "How to conciliate conflicting users' interests for different collective, ubiquitous and context-aware applications?" in *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*. IEEE, 2010, pp. 288–291.

[16] P. Carreira, S. Resendes, and A. C. Santos, "Towards automatic conflict detection in home and building automation systems," *Pervasive and Mobile Computing*, vol. 12, pp. 37–57, 2014.

[17] E. Göynügür, S. Bernardini, G. de Mel, K. Talamadupula, and M. Şensoy, "Policy conflict resolution in iot via planning," in *Canadian Conference on Artificial Intelligence*. Springer, 2017, pp. 169–175.