A Novel Spatial–Temporal Specification-Based Monitoring System for Smart Cities

Meiyi Ma, Ezio Bartocci, Eli Lifland, John A. Stankovic, *Life Fellow, IEEE*, and Lu Feng, *Member, IEEE*

***Abstract*—With the development of the Internet of Things, millions of sensors are being deployed in cities to collect real-time data. This leads to a need for checking city states against city requirements at runtime. In this article, we develop a novel spatial–temporal specification-based monitoring system for smart cities. We first describe a study of over 1000 smart city requirements, some of which cannot be specified using the existing logic, such as the signal temporal logic (STL) and its variants. To tackle this limitation, we develop spatial aggregation STL (SaSTL)—a novel spatial aggregation STL—for the efficient runtime monitoring of safety and performance requirements in smart cities. We develop two new logical operators in SaSTL to augment STL for expressing spatial aggregation and spatial counting characteristics that are commonly found in real city requirements. We define the Boolean and quantitative semantics for SaSTL in support of the analysis of city performance across different periods and locations. We also develop efficient monitoring algorithms that can check the SaSTL requirement in parallel over multiple data streams (e.g., generated by multiple sensors distributed spatially in a city). Additionally, we build an SaSTL-based monitoring tool to support decision making of different stakeholders to specify and runtime monitor their requirements in smart cities. We evaluate our SaSTL monitor by applying it to three case studies with large-scale real city sensing data (e.g., up to 10 000 sensors in one study). The results show that SaSTL has a much higher coverage expressiveness than other spatial–temporal logics, and with a significant reduction of computation time for monitoring requirements. We also demonstrate that the SaSTL monitor improves the safety and performance of smart cities via simulated experiments.**

***Index Terms*—Runtime verification, signal temporal logic (STL), smart cities.**

I. Introduction

SMART cities are emerging around the world. Examples include Chicago's Array of Things project [1], IBM's Rio de Janeiro Operations Center [2], and Cisco's Smart+Connected Operations Center [3], just to name a few. Smart cities utilize a vast amount of data and smart services to enhance the safety, efficiency, and performance of city operations [4]. There is a need for monitoring city states in real time to ensure safety and performance requirements [5]. If a requirement violation is detected by the monitor, the city operators and smart service providers can take actions to change the states, such as improving traffic performance, rejecting unsafe actions, sending alarms to police, etc. The key *challenges* of developing such a monitor include how to use an expressive machine-understandable language to specify smart city requirements, and how to efficiently monitor requirements that may involve multiple sensor data streams (e.g., some requirements are concerned with thousands of sensors in a smart city).

Previous works [6]–[9] have proposed solutions to monitor smart cities using formal specification languages and their monitoring machinery. One of the latest works, CityResolver [10] uses the signal temporal logic (STL) [11] to support the specification-based monitoring of safety and performance requirements of smart cities. However, STL is not expressive enough to specify smart city requirements concerning *spatial* information such as "the average noise level within 1 km of all elementary schools should always be less than 50 dB." There are some existing spatial extensions of STL (e.g., SSTL [12], SpaTeL [9], and STREL [13], [14], see [15] for a recent tutorial), which can express requirements such as "there should be no traffic congestion on all the roads in the northeast direction." But they are not expressive enough to specify requirements like "there should be no traffic congestion on all the roads on average," or "on 90% of the roads," which require the aggregation and counting of signals in the spatial domain. To tackle these challenges and limitations, we develop a novel spatial aggregation STL (SaSTL), which extends STL with two new logical operators for expressing spatial aggregation and spatial counting characteristics, which we demonstrate are commonly found in real city requirements. More specifically, this article has the following major contributions.

1) To the best of our knowledge, this is the first work studying and annotating over 1000 real smart city requirements across different service domains to identify the gap of expressing smart city requirements with existing formal specification languages. As a result, we found that aggregation and counting signals in the spatial domain (e.g., for representing sensor signals distributed spatially in a smart city) are extremely important for specifying and monitoring city requirements.
2) Drawing on the insights from our requirements study, we develop a new specification language SaSTL, which extends STL with a *spatial aggregation* operator and a *spatial counting* operator. SaSTL can be used to specify

Manuscript received December 8, 2020; revised February 20, 2021; accepted March 17, 2021. Date of publication March 31, 2021; date of current version July 23, 2021. This work was supported in part by the National Science Foundation under Grant CCF-1942836 and Grant CNS-1952096, and in part by the Austrian FFG-Funded IoT4CPS Project at TU Wien. *(Corresponding author: Meiyi Ma.)*

Meiyi Ma, Eli Lifland, John A. Stankovic, and Lu Feng are with the Department of Computer Science, University of Virginia, Charlottesville, VA 22904 USA (e-mail: meiyi@virginia.edu).

Ezio Bartocci is with the Faculty of Informatics, TU Wien, 1040 Vienna, Austria.

Digital Object Identifier 10.1109/JIOT.2021.3069943

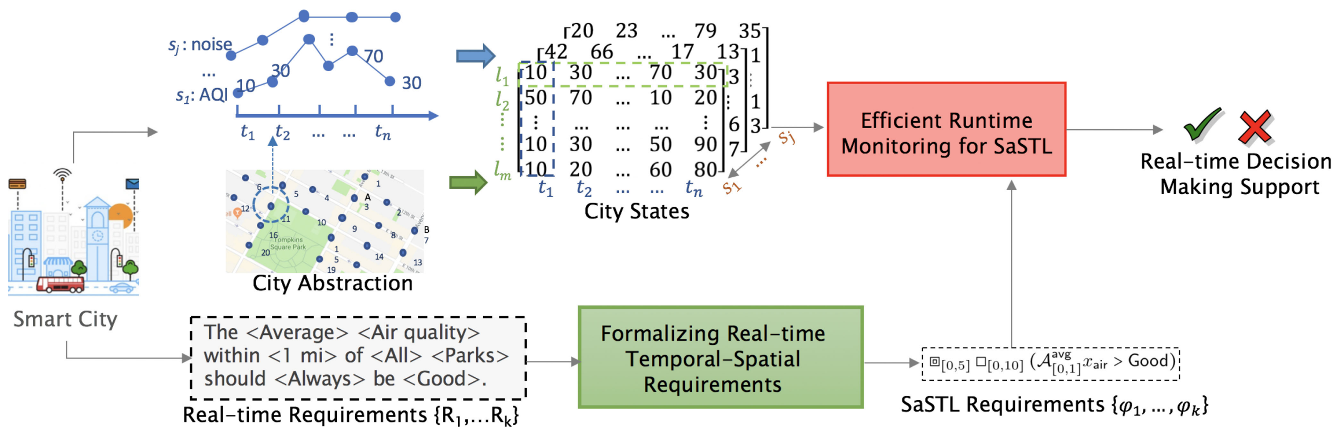

s_j: noise
...
s_1: AQI
10
30
70
30
t_1 t_2 ... ... t_n
City States
City Abstraction
Smart City
Efficient Runtime Monitoring for SaSTL
Real–time Decision Making Support
The <Average> <Air quality> within <1 mi> of <All> <Parks> should <Always> be <Good>.
Real–time Requirements {R1,...Rk}
Formalizing Real–time Temporal-Spatial Requirements
$\boxdot_{[0,5]} \square_{[0,10]} (\mathcal{A}^{avg}_{[0,1]} x_{air} > \text{Good})$
SaSTL Requirements $\{\varphi_1, \ldots, \varphi_k\}$

Fig. 1. Framework for runtime monitoring of real-time city requirements.

the Point of Interests (PoIs), physical distance, spatial relations of the PoIs and sensors, aggregation of signals over locations, degree/percentage of satisfaction, and the temporal elements in a very flexible spatial–temporal scale. We define the Boolean and quantitative semantics with theoretical proofs.

3) We compare SaSTL with some existing specification languages and show that SaSTL has a much higher coverage expressiveness (95%) than STL (18.4%), SSTL (43.1%) or STREL (43.1%) over 1000 real city requirements.
4) We develop novel and efficient monitoring algorithms for SaSTL. In particular, we present two new methods to speed up the monitoring performance: a) dynamically prioritizing the monitoring based on cost functions assigned to nodes of the syntax tree and b) parallelizing the monitoring of spatial operators among multiple locations and/or sensors.
5) We evaluate the SaSTL monitor by applying it to monitoring real city data collected from Chicago and Aarhus. The results show that SaSTL monitor has the potential to help identify safety violations and support the city managers and citizens to make decisions. We also evaluate the SaSTL monitor on a third case study of conflict detection and resolution among smart services in simulated New York City with large-scale real sensing data (e.g., up to 10 000 sensors used in one requirement). Results of our simulated experiments show that SaSTL monitor can help improve the city's performance (e.g., 21.1% on the environment and 16.6% on public safety), with a significant reduction of computation time compared with previous approaches.
6) We develop an SaSTL monitoring tool that can support decision making of different stakeholders in smart cities. The tool allows users (e.g., the city decision maker and citizens) without any formal method background to specify city requirements and monitor city performance easily.

This article is an extended version of [16]. We extend with the following new contributions. First, we add new quantitative semantics and monitoring algorithms, with new theorems of soundness and correctness in Section III. Compared to the conference version (the Boolean semantics), the new quantitative semantics presents the monitoring results with real values, and better supports decision makers to compare the satisfaction/violation degrees between different options. Second, we develop new monitoring algorithms for the proposed quantitative semantics and improve the monitoring algorithms for the new spatial operators in Section IV. Third, we develop a monitoring tool to support monitoring and decision making using SaSTL in smart cities in Section VI. The tool also provides a way for nonexpert users to input requirements in the English language. Then, the tool translates the requirements to the SaSTL formal specification automatically for monitoring. Fourth, we extend the evaluation with a new city scenario using real data from Aarhus, Denmark in Section VII. The results show that the SaSTL monitor has the potential to help identify safety violations and support city managers and citizens to make decisions. Finally, we elaborate with more discussions on how to apply the SaSTL monitor in smart cities and extend the related work.

II. Approach Overview

Fig. 1 shows an overview of our SaSTL runtime monitoring framework for smart cities. We envision that such a framework would operate in a smart city's central control center (e.g., IBM's Rio de Janeiro Operations Center [2] or Cisco's Smart+Connected Operations Center [3]) where sensor data about city states across various locations are available in real time. The framework would monitor city states and check them against a set of smart city requirements at runtime. The monitoring results would be presented to city managers to support decision making. The framework makes abstractions of city states in the following way. The framework formalizes a set of smart city requirements (see Section III) to some machine checkable SaSTL formulas (see Section IV). Different data streams (e.g., CO emission and noise level) over temporal and spatial domains can be viewed as a 3-D matrix. For any signal s_j in signal domain S, each row is a time-series data at one location and each column is a set of data streams from all locations at one time. Next, the efficient real-time monitoring for SaSTL verifies the states with the requirements and outputs the Boolean satisfaction to the decision makers, who would take actions to resolve the violation. To support decision making in real time, we improve the efficiency of the monitoring algorithm in Section V. We implement the SaSTL runtime monitoring tool following this framework for

TABLE I
EXAMPLES OF CITY REQUIREMENTS FROM DIFFERENT DOMAINS (THE KEY ELEMENTS OF A REQUIREMENT ARE HIGHLIGHTED AS, TEMPORAL, SPATIAL, AGGREGATION, ENTITY, CONDITION, AND COMPARISON)

Domain	Example
Transportation	Limits vehicle idling to one minute adjacent to any school, pre-K to 12th grade, public or private, in the City of New York [17].
	The engine, power and exhaust mechanism of each motor vehicle shall be equipped, adjusted and operated to prevent the escape of a trail of visible fumes or smoke for more than ten (10) consecutive seconds [18].
	Prohibit sight-seeing buses from using all bus lanes between the hours of 7:00 a.m. and 10:00 a.m. on weekdays [19].
Energy	Operate the system to maintain zone temperatures down to 55°F or up to 85°F [20].
	The total leakage shall be less than or equal to 4 cubic feet per minute per 100 square feet of conditioned floor area [21].
Environment	LA Sec 111.03 minimum ambient noise level table: ZONE M2 and M3 – DAY : 65 dB(A) NIGHT : 65 dB(A) [22].
	The total amount of HCHO emission should be less than 0.1mg per m^3 within an hour, and the total amount of PM10 emission should be less than 0.15 mg per m^3 within 24 hours [23].
Emergency	NYC Authorized emergency vehicles may disregard 4 primary rules regarding traffic [24].
	At least one ambulance should be equipped per 30,000 population (counted by area) to obtain the shortest radius and fastest response time [25].
Public Safety	Security staff shall visit at least once per week in public schools [26].

TABLE II
KEY ELEMENTS OF CITY REQUIREMENTS AND STATISTICAL RESULTS FROM 1000 REAL CITY REQUIREMENTS

Element	Form	Number	Example
Temporal	Dynamic Deadline	77	limit ... to one minute
	Static Deadline	98	at least once a week
	Interval	168	from 8am to 10am; within 24 hours;
	Default	657	The noise (always) should not exceed 50dB.
Spatial	**PoIs/Tags**	**801**	school area; all parks;
	Distance	**650**	Nearby
	Default	154	(everywhere) ; (all) locations
Aggregation	**Count, Sum**	**256**	in total; x out of N locations; %;
	Average	**196**	per m^2;
	Max, Min	**67**	highest/lowest value
Entity	Subject	1000	air quality; Buses;
Comparison	Value comparison	836	More than, less than
	Boolean	388	Street is blocked; should
	Not	456	It is unlawful/prohibited...
Condition	Until	24	keep... until the street is not blocked.
	If/Except	44	If rainy, the speed limit...

city experts without any formal methods background (see Section VI). We describe more details of the framework in the following sections.

III. Analysis of Real City Requirements

To better understand real city requirements, we conduct a requirement study. We collect and statistically analyze 1000 quantitatively specified city requirements (e.g., standards, regulations, city codes, and laws) across different application domains, including energy, environment, transportation, emergency, and public safety from over 70 cities (e.g., New York City, San Francisco, Chicago, Washington D.C., Beijing, etc.) around the world. Some examples of these city requirements are highlighted in Table I. We identify the key required features to have in a specification language and its associated use in a city runtime monitor. The summarized statistical results of the study and key elements we identified (i.e., temporal, spatial, aggregation, entity, comparison, and condition) are shown in Table II.

Temporal: Most of the requirements include a variety of temporal constraints, e.g., a static deadline, a dynamic deadline, or time intervals. In many cases (65.7%), the temporal information is not explicitly written in the requirement, which usually means it should be "always" satisfied. In addition, city requirements are highly real-time driven. In over 80% requirements, cities are required to detect requirement violations at runtime. It indicates a high demand for runtime monitoring.

Spatial: A requirement usually specifies its spatial range explicitly using the PoIs (80.1%), such as "park," "xx school," along with a distance range (65%). One requirement usually points to a set of places (e.g., all the schools). Therefore, it is very important for a formal language to be able to specify the spatial elements across many locations within the formula, rather than one formula for each location.

We also found that the city requirements specify a very large spatial scale. Different from the requirements of many other CPS, requirements from smart cities are highly spatial-specific and usually involve a very large number of locations/sensors. For example, the first requirement in Table I specifies a vehicle idling time "adjacent to any school, pre-K to 12th grade in the City of New York." There are about 2000 pre-K to 12th schools, even counting 20 street segments nearby each school, there are 40 000 data streams to be monitored synchronously. An efficient monitoring is highly demanded.

Aggregation: In 51.9% of cases, requirements are specified on the aggregated signal over an area, such as "the total amount," "average$\cdots$ per 100 square feet," "up to four vending vehicles in any given city block," "at least 20% of travelers from all entrances should$\cdots$," etc. Therefore, aggregation is a key feature for the specification language.

IV. Formalizing Temporal–Spatial Requirements

SaSTL extends STL with two spatial operators: 1) a *spatial aggregation* operator and 2) a *neighborhood counting* operator. Spatial aggregation enables combining (according to a chosen operation) measurements of the same type (e.g., environmental temperature), but taken from different locations. The use of this operator can be suitable in requirements, where it is necessary to evaluate the average, best, or worst value of a signal measurement in an area close to the desired location. The neighborhood counting operator allows measuring the number/percentage of neighbors of a location that satisfies a certain requirement.

A. SaSTL Syntax

We define a multidimensional *spatial–temporal signal* as $\omega : \mathbb{T} \times L \to \{\mathbb{R} \cup \{\perp\}\}^n$, where $\mathbb{T} = \mathbb{R}_{\geq 0}$ represents the continuous time and L is the set of locations. We define $X = \{x_1, \ldots, x_n\}$ as the set of variables for each location. Each variable can assume a real value $v \in \mathbb{R}$ or is undefined for a particular location ($x_i = \perp$). We denote by $\pi_{x_i}(\omega)$ as the projection of ω on its component variable $x_i \in X$. We define $P = \{p_1, \ldots, p_m\}$ a set of propositions (e.g., $\{\mathsf{School}, \mathsf{Street}, \mathsf{Hospital}, \ldots\}$) and L a labeling function $L : L \to 2^P$ that assigns for each location the set of the propositions that are true in that location.

A *weighted undirected graph* is a tuple $G = (L, E, \eta)$, where L is a finite nonempty set of nodes representing locations, $E \subseteq L \times L$ is the set of edges connecting nodes, and $\eta : E \to \mathbb{R}_{\geq 0}$ is a cost function over edges. We define the *weighted distance* between two locations $l, l' \in L$ as

$$d(l, l') := \min\left\{\sum_{e\in\sigma} \eta(e) | \sigma \text{ is a path between } l \text{ and } l'\right\}.$$

Then, we define the spatial domain $\mathcal{D}$ as

$$\mathcal{D} := ([d_1, d_2], \psi)$$
$$\psi := \top | p | \neg\, \psi | \psi \ \vee\ \psi$$

where $[d_1, d_2]$ defines a spatial interval with $d_1 < d_2$ and $d_1, d_2 \in \mathbb{R}$, and ψ specifies the property over the set of propositions that must hold in each location. Intuitively, it draws two circles with radius $r_1 = d_1$ and $r_2 = d_2$, and the locations $l \models \psi$ between these two circles are selected. In particular, $\mathcal{D} = ([0, +\infty), \top)$ indicates the whole spatial domain. We denote $L^l_{([d_1,d_2],\psi)} := \{l' \in L | 0 \leq d_1 \leq d(l, l') \leq d_2$ and $\mathcal{L}(l') \models \psi\}$ as the set of locations at a distance between d_1 and d_2 from l for which $\mathcal{L}(l')$ satisfies ψ. We denote the set of nonnull values for signal variable x at time point t location l over locations in $L^l_{\mathcal{D}}$ by

$$\alpha^x_{\mathcal{D}}(\omega, t, l) := \{\pi_x(\omega)[t, l'] | l' \in L^l_{\mathcal{D}} \text{ and } \pi_x(\omega)[t, l'] \neq \perp\}.$$

We define a set of operations $\text{op}(\alpha^x_{\mathcal{D}}(\omega, t, l))$ for op $\in$ $\{\max, \min, \text{sum}, \text{avg}\}$ when $\alpha^x_{\mathcal{D}}(\omega, t, l) \neq \emptyset$ that computes the maximum, minimum, summation, and average of values in the set $\alpha^x_{\mathcal{D}}(\omega, t, l)$, respectively. To be noted, Graph G and its weights between nodes are constructed flexibly based on the property of the system. For example, we can build a graph with fully connected sensor nodes and their Euclidean distance as the weights when monitoring the air quality in a city; or we can also build a graph that only connects the street nodes when the two streets are contiguous and apply Manhattan distance. It does not affect the syntax and semantics of SaSTL.

The syntax of SaSTL is given by

$$\varphi := x \sim c | \neg\varphi | \varphi_1 \wedge \varphi_2 | \varphi_1 \mathcal{U}_I \varphi_2 | \mathcal{A}^{\text{op}}_{\mathcal{D}} x \sim c | \mathcal{C}^{\text{op}}_{\mathcal{D}} \varphi \sim c$$

where $x \in X$, $\sim \in \{<, \leq\}$, $c \in \mathbb{R}$ is a constant, $I \subseteq \mathbb{R}_{>0}$ is a real positive dense time interval, and $\mathcal{U}_I$ is the *bounded until* temporal operators from STL. The *always* (denoted $\Box$) and *eventually* (denoted $\Diamond$) temporal operators can be derived the same way as in STL, where $\Diamond\varphi \equiv \mathsf{true}\ \mathcal{U}_I \varphi$ and $\Box\varphi \equiv \neg\Diamond\neg\varphi$.

In SaSTL, we define a set of spatial *aggregation* operators $\mathcal{A}^{\text{op}}_{\mathcal{D}} x \sim c$ for op $\in \{\max, \min, \text{sum}, \text{avg}\}$ that evaluate the aggregated product of traces $\text{op}(\alpha^x_{\mathcal{D}}(\omega, t, l))$ over a set of locations $l \in L^l_{\mathcal{D}}$. We also define a set of new spatial *counting* operators $\mathcal{C}^{\text{op}}_{\mathcal{D}} \varphi \sim c$ for op $\in \{\max, \min, \text{sum}, \text{avg}\}$ that counts the satisfaction of traces over a set of locations. More precisely, we define $\mathcal{C}^{\text{op}}_{\mathcal{D}} \varphi = \text{op}(\{g((\omega, t, l') \models \varphi) | l' \in L^l_{\mathcal{D}}\})$, where $g((\omega, t, l) \models \varphi)) = 1$ if $(\omega, t, l) \models \varphi$; otherwise, $g((\omega, t, l) \models \varphi)) = 0$. From the new *counting* operators, we also derive the *everywhere* operator as $\boxdot_{\mathcal{D}} \varphi \equiv \mathcal{C}^{\min}_{\mathcal{D}} \varphi > 0$, and *somewhere* operator as $\diamondsuit_{\mathcal{D}} \varphi \equiv \mathcal{C}^{\max}_{\mathcal{D}} \varphi > 0$. In addition, $\mathcal{C}^{\text{sum}}_{\mathcal{D}} \varphi$ specifies the total number of locations that satisfy φ and $\mathcal{C}^{\text{avg}}_{\mathcal{D}} \varphi$ specifies the percentage of locations satisfying φ.

We now illustrate how to use SaSTL to specify various city requirements, especially for the spatial aggregation and spatial counting, and how important these operators are for the smart city requirements using examples below.

Example 1 (Spatial Aggregation): Assume that we have a requirement, "the average noise level in the school area (within 1 km) in New York City should always be less than 50 dB and the worst should be less than 80 dB in the next 3 h" is formalized as, $\boxdot_{([0,+\infty),\mathsf{School})} \Box_{[0,3]}((\mathcal{A}^{\text{avg}}_{([0,1],\top)} x_{\mathsf{Noise}} < 50) \wedge (\mathcal{A}^{\max}_{([0,1],\top)} x_{\mathsf{Noise}} < 80))$. $([0, +\infty), \mathsf{School})$ selects all the locations labeled as "school" within the whole New York city $([0, +\infty))$ (predefined by users). $\Box_{[0,3]}$ indicates this requirement is valid for the next three hours. $(\mathcal{A}^{\text{avg}}_{([0,1],\top)} x_{\mathsf{Noise}} < 50) \wedge (\mathcal{A}^{\max}_{([0,1],\top)} x_{\mathsf{Noise}} < 80)$ calculates the average and maximal values in 1 km for each "school," and compares them with the requirements, i.e., 50 dB and 80 dB.

Without the spatial aggregation operators, STL and its extended languages cannot specify this requirement. First, they are not able to first dynamically find all the locations labeled as "school." To monitor the same spatial range, users have manually get all traces from schools, and then repeatedly apply this requirement to each located sensor within 1 km of a school and do the same for all schools. More importantly, STL and its extended languages could not specify "average" or "worst" noise level. Instead, it only monitors each single value, which is prone to noises and outliers and thereby causes inaccurate results.

Example 2 (Spatial Counting): A requirement that "at least 90% of the streets, the particulate matter (PMx) emission should not exceed *Moderate* in 2 h" is formalized as $\mathcal{C}^{\text{avg}}_{([0,+\infty),\mathsf{Street})}(\Box_{[0,2]}(x_{\mathsf{PMx}} < \mathsf{Moderate})) > 0.9$. $\mathcal{C}^{\text{avg}}_{([0,+\infty),\mathsf{Street})} \varphi > 0.9$ represents the percentage of satisfaction is larger than 90%. Specifying the percentage of satisfaction is very common and important among city requirements.

B. SaSTL Semantics

We define the SaSTL semantics as the *satisfiability relation* $(\omega, t, l) \models \varphi$, indicating that the spatiotemporal signal ω satisfies a formula φ at the time point t in location l when $\pi_v(\omega)[t, l] \neq \perp$ and $\alpha^x_{\mathcal{D}}(\omega, t, l) \neq \emptyset$. We define that

$(\omega, t, l) \models \varphi$ if $\pi_v(\omega)[t, l] = \bot$

$$\begin{aligned}
(\omega, t, l) &\models x \sim && \Leftrightarrow \pi_x(\omega)[t, l] \sim c\\
(\omega, t, l) &\models \neg\varphi && \Leftrightarrow (\omega, t, l) \not\models \varphi\\
(\omega, t, l) &\models \varphi_1 \wedge \varphi_2 && \Leftrightarrow (\omega, t, l) \models \varphi_1 \text{ and } (\omega, t, l) \models \varphi_2\\
(\omega, t, l) &\models \varphi_1 \mathcal{U}_I \varphi_2 && \Leftrightarrow \exists t' \in (t + I) \cap \mathbb{T} : (\omega, t', l) \models \varphi_2\\
& && \quad \text{and } \forall t'' \in (t, t'), (\omega, t'', l) \models \varphi_1\\
(\omega, t, l) &\models \mathcal{A}^{\text{op}}_{\mathcal{D}} x \sim c && \Leftrightarrow \text{op}(\alpha^x_{\mathcal{D}}(\omega, t, l)) \sim c\\
(\omega, t, l) &\models \mathcal{C}^{\text{op}}_{\mathcal{D}} \varphi \sim c && \Leftrightarrow \text{op}(\{g((\omega, t, l') \models \varphi) | l' \in L^l_{\mathcal{D}}\}) \sim c
\end{aligned}$$

where, for counting operator $(\omega, t, l) \models \mathcal{C}^{\text{op}}_{\mathcal{D}} \varphi \sim c$, the valid ranges for c are $c \in [0, 1)$ when op = sum/min, and $c \in [0, N]$ when op = sum/min. Otherwise, (e.g., $c < 0$), the requirement is trivially satisfied or violated.

Example 3: Following Example 1, checking the city states with a requirement $\boxdot_{([0,+\infty),\text{Hospital})} \square_{[0,5]}((\mathcal{A}^{\text{avg}}_{([0,500],\top)} x_{\text{AQI}} < 50) \wedge (\mathcal{A}^{\max}_{([0,500],\top)} x_{\text{AQI}} < 80))$, to start with, assuming we have the AQI level data from a number of sensors within 500 m of one of the hospital, the sensor readings in 5 h as, $\{[51, \ldots, 11], [80, \ldots, 30], \ldots, [40, \ldots, 30]\}$, $\varphi_t = (\mathcal{A}^{\text{avg}}_{([0,500],\top)]} x_{\text{AQI}} < 50) \wedge (\mathcal{A}^{\max}_{([0,500],\top)} x_{\text{AQI}} < 80)$, then, we check φ_t for this hospital at each time, at $t = 1$, $\text{avg}(51, \ldots, 40) > 50 \wedge \max(51, \ldots, 40) < 80$, thus, $\varphi_{t1} = \mathit{False}$, at $t = 2$, $\text{avg}(49, \ldots, 20) < 50 \wedge \max(49, \ldots, 20) > 80$, thus, $\varphi_{t1} = \mathit{False}$, $\ldots$, at $t = 5$, $\text{avg}(11, \ldots, 30) < 50 \wedge \max(11, \ldots, 30) < 80$, thus, $\varphi_{t1} = \mathit{True}$.

Thus, we have $\square_{[0,5]}\varphi_t = \mathit{False}$.

Next, the monitor checks all qualified hospitals the same way and reaches the final results $\boxdot_{([0,+\infty),\text{Hospital})} \square_{[0,5]} ((\mathcal{A}^{\text{avg}}_{([0,500],\top)} x_{\text{AQI}} < 50) \wedge (\mathcal{A}^{\max}_{([0,500],\top)}\ x_{\text{AQI}} < 80)) = \mathit{False}$.

In a real scenario, the monitor algorithm can also decide to terminate the monitor and return the False result when at $t = 1$, because the always operator returns False as long as a one-time violation occurs. Similarly, the everywhere operator will also return False when the first hospital violates the requirement.

Definition 1 (Quantitative Semantics): Let $x > c$ be a numerical predicate, we then define the robustness degree (i.e., the quantitative satisfaction) function $\rho(\varphi, \omega, t, l)$ for an SaSTL formula over a spatial–temporal signal ω as

$$\begin{aligned}
\rho(x \sim c, \omega, t, l) &= \pi_x(\omega)[t, l] - c\\
\rho(\neg\varphi, \omega, t, l) &= -\rho(\varphi, \omega, t, l)\\
\rho(\varphi_1 \vee \varphi_2, \omega, t, l) &= \max\{\rho(\varphi_1, \omega, t, l), \rho(\varphi_2, \omega, t, l)\}\\
\rho(\varphi_1 \mathcal{U}_I \varphi_2, \omega, t, l) &= \sup_{t' \in (t+I) \cap \mathbb{T}} (\min\{\rho(\varphi_2, \omega, t', l),\\
&\qquad \inf_{t'' \in [t, t']} (\rho(\varphi_1, \omega, t'', l))\})
\end{aligned}$$

$$\rho(\mathcal{A}^{\text{op}}_{\mathcal{D}} x \sim c, \omega, t, l) = \begin{cases} \frac{\text{sum}(\alpha^x_{\mathcal{D}}(\omega, t, l)) - c}{|\alpha^x_{\mathcal{D}}(\omega, t, l)|}, & \text{op= sum}\\ \text{op}(\alpha^x_{\mathcal{D}}(\omega, t, l)) - c, & \text{op} \in \{\text{max, min, avg}\} \end{cases}$$

$$\rho(\mathcal{C}^{\text{op}}_{\mathcal{D}} \varphi \sim c, \omega, t, l)$$

$$= \begin{cases} \max_{l' \in L^l_{\mathcal{D}}}\{\rho(\varphi, \omega, t, l')\}, & \text{op = max}\\ \min_{l' \in L^l_{\mathcal{D}}}\{\rho(\varphi, \omega, t, l')\}, & \text{op = min}\\ \delta(\lceil c \rceil, \{\rho(\varphi, \omega, t, l') | l' \in L^l_{\mathcal{D}}\}), & \text{op = sum}\\ \delta(\lceil c \times |L^l_{\mathcal{D}}| \rceil, \{\rho(\varphi, \omega, t, l') | l' \in L^l_{\mathcal{D}}\}), & \text{op = avg} \end{cases}$$

where we define *function*(k, S) as a function that returns the kth smallest number of set S, $|S| > 0$, and $0 \leq k \leq |S|$. For $\mathcal{C}^{\text{op}}_{\mathcal{D}} \varphi \sim c$, when op = sum, it requires that there are at least $\lceil c \rceil$ locations that satisfy φ; thus, we denote the $\lceil c \rceil$th smallest robustness value from $\{\rho(\varphi, \omega, t, l') | l' \in L^l_{\mathcal{D}}\}$ as the robustness value of this formula. $[c]$ indicates the smallest integer that is larger than or equal to c. Similarly, when op = avg, the formula is converted as there are at least $\lceil c \times |L^l_{\mathcal{D}}| \rceil$ locations that satisfy φ; thus, we denote the $\lceil c \times |L^l_{\mathcal{D}}| \rceil$th smallest robustness value from $\{\rho(\varphi, \omega, t, l') | l' \in L^l_{\mathcal{D}}\}$ as the robustness value of this formula. Same as the Boolean semantics, the valid ranges for c are $c \in [0, 1)$ when op = sum/min, and $c \in [0, N]$ when op = sum/min. Otherwise (e.g., $c < 0$), the requirement is trivially satisfied or violated.

Example 4: Assuming we have data (1, 2, 3), (2, 3, 4), (4, 5, 7) from three locations satisfying $\mathcal{D}$, thus:

1) $\rho(\mathcal{C}^{\max}_{\mathcal{D}}(\square_{[0,2]}(x > 5)) > 0) = \rho(\mathcal{C}^{\max}_{\mathcal{D}}(\{-4, -3, 2\}) > 0) = 2$;
2) $\rho(\mathcal{C}^{\min}_{\mathcal{D}}(\square_{[0,2]}(x > 5)) > 0) = \rho(\mathcal{C}^{\min}_{\mathcal{D}}(\{-4, -3, 2\}) > 0) = -4$;
3) $\rho(\mathcal{C}^{\text{sum}}_{\mathcal{D}}(\square_{[0,2]}(x > 5)) > 1) = \rho(\mathcal{C}^{\text{sum}}_{\mathcal{D}}(\{-4, -3, 2\}) > 1) = -3$;
4) $\rho(\mathcal{C}^{\text{avg}}_{\mathcal{D}}(\square_{[0,2]}(x > 5)) > 0.2) = \rho(\mathcal{C}^{\text{avg}}_{\mathcal{D}}(\{-4, -3, 2\}) > 0.2) = 2$.

The quantitative semantics of SaSTL inherit the two fundamental properties of STL, i.e., soundness and correctness. We give the formal definitions below.

Theorem 1 (Soundness): Let φ be an STL formula, ω a trace and t a time

$$\begin{aligned}
\rho(\varphi, \omega, t, l) > 0 &\Rightarrow (\omega, t, l) \models \varphi\\
\rho(\varphi, \omega, t, l) < 0 &\Rightarrow (\omega, t, l) \not\models \varphi.
\end{aligned}$$

Second, if ω satisfies φ at time t, any other trace ω' whose point-wise distance from ω is smaller than $\rho(\varphi, \omega, t, l)$ also satisfies φ at time t.

Theorem 2 (Correctness): Let φ be an STL formula, ω and ω' traces over the same time and spatial domains, and $t, l \in \text{dom}(\varphi, \omega)$, then

$$(\omega, t, l) \models \varphi \text{ and } ||\omega - \omega'||_\infty < \rho(\varphi, \omega, t, l) \Rightarrow (\omega', t, l) \models \varphi.$$

In summary, the qualitative value indicates if the signal (i.e., city data) satisfies the requirement. The quantitative value indicates the satisfaction or dissatisfaction degree. If it is larger than or equal to 0, it means that the requirement is satisfied. The larger the value, the more the requirement is satisfied. On the contrary, if the value is smaller than 0, it means the requirement is not satisfied. The smaller the value, the more the requirement is dissatisfied.

V. Efficient Monitoring for SaSTL

In this section, we first present both Boolean and quantitative monitoring algorithms for SaSTL, then describe two optimization methods to speed up the monitoring performance.

A. Monitoring Algorithms for SaSTL

The *inputs* of the monitor are the SaSTL requirements φ (including time t and location l), a weighted undirected graph G, and the temporal-spatial data ω. In smart

Algorithm 1: SaSTL Quantitative Monitoring Algorithm MonitorQ(φ, ω, t, l, G)

```
Input: SaSTL Requirement $\varphi$, Signal $\omega$, Time $t$, Location $l$, weighted
       undirected graph $G$
Output: Satisfaction Value $\rho$
begin
    switch $\varphi$ do
        Case $x \sim c$
            return $\pi_x(\omega)[t, l] - c$;
        Case $\neg\varphi$
            return- MonitorQ($\varphi, \omega, t, l, G$);
        Case $\varphi_1 \wedge \varphi_2$
            return min(MonitorQ($\varphi_1, \omega, t, l, G$),
            MonitorQ($\varphi_2, \omega, t, l, G$));
        Case $\varphi_1 \mathcal{U}_I \varphi_2$
            Real $v := -\infty$
            for $t' \in (t + I) \cap \mathbb{T}$ do
                $v' :=$ MonitorQ($\varphi_2, \omega, t', l, G$)
                for $t'' \in [t, t']$ do
                    $v' := \min\{v'$, MonitorQ($\varphi_2, \omega, t'', l, G$)$\}$
                end
                $v = \max\{v, v'\}$
            end
            return v;
        Case $\mathcal{A}_{\mathcal{D}}^{op} x \sim c$ ;                         ▷ See Alg. 2.
            return AggregateQ($x, c, op, \mathcal{D}, t, l, G$);
        Case $\mathcal{C}_{\mathcal{D}}^{op} \varphi \sim c$ ;                         ▷ See Alg. 3.
            return CountingNeighboursQ($\varphi, c, op, \mathcal{D}$,
            $t, l, G$);
    end
end
```

Algorithm 2: AggregateQ($x, op, \mathcal{D}, \omega, t, l, G$)

```
begin
    Real v := 0; n := 0;
    if $op$ == "$min$" then $v := \infty$;
    if $op$ == "$max$" then $v := -\infty$;
    $L_{\mathcal{D}}^{l}$ := deScan($l, G, \mathcal{D}$)
    for $l' \in L_{\mathcal{D}}^{l}$ do
        if $op \in \{min, max, sum\}$ then
            $v :=$ op($v, \pi_x(\omega)[t, l']$);
        end
        if $op$ == "$avg$" then
            $v :=$ sum($v, \pi_x(\omega)[t, l']$);
        end
        $n := n + 1$
    end
    if $n == 0$ then return $\infty$;
    if $op$ == "$avg$" $\wedge n \neq 0$ then return $v/n - c$ ;
    if $op$ == "$sum$" $\wedge n \neq 0$ then return $(v - c)/n$ ;
    else return $v - c$;
end
```

Algorithm 3: CountingNeighboursQ($x, op, \mathcal{D}, \omega, t, l, G$)

```
begin
    Real $n := 0$, List $s := Null$;
    $L_{\mathcal{D}}^{l}$ := deScan($l, G, \mathcal{D}$)
    for $l' \in L_{\mathcal{D}}^{l}$ do
        s.add(Monitor($\varphi, \omega, t, l', G$))
        $n := n + 1$
    end
    if $n == 0$ then return $\infty$;
    else
        switch $op$ do
            Case max
                return $s$.max()
            Case min
                return $s$.min()
            Case $sum$
                return $s$.max(round($c$))
            Case $avg$
                return $s$.max(round($c \times n$))
        end
    end
end
```

cities, the data on city states are collected continuously or periodically.

For the Boolean monitoring algorithm, the *output* for each requirement is a Boolean value indicating whether the requirement is satisfied or not. For the quantitative monitoring algorithm (Algorithm 1), the *output* for each requirement is a number indicating the satisfaction degree of the requirement. To start with, the monitoring algorithm parses φ to subformulas and calculates the satisfaction for each operation recursively. We derived operators $\Box$ and $\Diamond$ from $\mathcal{U}_I$, and operators $\boxdot$ and $\diamonddot$ from $\mathcal{C}_{\mathcal{D}}^{\text{op}} \sim c$, so we only show the algorithms for $\mathcal{U}_I$ and $\mathcal{C}_{\mathcal{D}}^{\text{op}} \sim c$.

We present the quantitative monitoring algorithms of the operators $\mathcal{A}_{\mathcal{D}}^{\text{op}}$ and $\mathcal{C}_{\mathcal{D}}^{\text{op}}$ in Algorithm 2 and Algorithm 3, respectively. We apply distributed parallel algorithm deScan() [27] to accelerate the process of searching locations that satisfy $\mathcal{D}$. As we can tell from the algorithms, essentially, $\mathcal{A}_{\mathcal{D}}^{\text{op}}$ calculates the aggregated values on the signal over a spatial domain, while $\mathcal{C}_{\mathcal{D}}^{\text{op}}$ calculates the aggregated results over spatial domain. For the quantitative monitoring algorithm (as presented in Algorithm 1), the *output* for each requirement is a robustness value indicating its satisfaction degree. Similar to the Boolean monitoring algorithm, the quantitative monitoring algorithm also parses φ to subformulas and calculates the satisfaction for each operation recursively.

The time complexity of monitoring the logical and temporal operators of SaSTL is the same as STL [28]. The time complexity to monitor classical logical operators or basic propositions, such as $\neg x$, $\wedge$, and $x \sim c$ is $O(1)$. The time complexity to monitor temporal operators, such as $\Box_I$, $\Diamond_I$, and $\mathcal{U}_I$ is $O(T)$, where T is the total number of samples within time interval I. In this article, we present the time complexity analysis for the spatial operators (Lemma 1) and the new SaSTL monitoring algorithm (Theorem 3). The total number of locations is denoted by n. We assume that the positions of the locations cannot change in time (a fixed grid). We can precompute all the distances between locations and store them in an array of range trees [29] (one range tree for each location). We further denote the monitored formula as ϕ, which can be represented by a syntax tree, and let $|\phi|$ denote the total number of nodes in the syntax tree (the number of operators).

Lemma 1 (Complexity of Spatial Operators): The time complexity to monitor at each location l at time t the satisfaction of a spatial operator, such as $\boxdot_{\mathcal{D}}$, $\diamonddot_{\mathcal{D}}$, $\mathcal{A}_{\mathcal{D}}^{\text{op}}$, and $\mathcal{C}_{\mathcal{D}}^{\text{op}}$ is $O(\log(n) + |L|)$ where L is the set of locations at distance within the range $\mathcal{D}$ from l.

Theorem 3: The time complexity of the SaSTL monitoring algorithm is upper bounded by $O(|\phi| \times T_{\max} \times (\log(n) + |L|_{\max}))$, where $T_{\max}$ is the largest number of samples of the intervals considered in the temporal operators of ϕ and $|L|_{\max}$

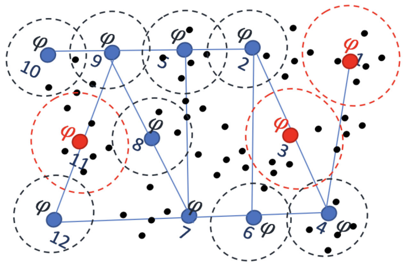


Fig. 2. Example of city abstracted graph. A requirement is $\boxdot_{([0,+\infty),\text{School})}\Box_{[a,b]}(\mathcal{A}^{\text{op}}_{([0,d],\top)}\varphi \sim c)$. (The large nodes represent the locations of PoIs, among which the red ones represent the schools, and blue ones represent other PoIs. The small black nodes represent the locations of data sources.)

is the maximum number of locations defined by the spatial temporal operators of ϕ.

B. Performance Improvement of SaSTL Parsing

To monitor a requirement, the first step is parsing the requirement to a set of sub formulas with their corresponding spatial–temporal ranges. Then, we calculate the results for the subformulas. The traditional parsing process of STL builds and calculates the syntax tree on the sequential order of the formula. It does not consider the complexity of each subformula. However, in many cases, especially with the PoIs specified in smart cities, checking the simpler propositional variable to quantify the spatial domain first can significantly reduce the number of temporal signals to check in a complicated formula. For example, the city abstracted graph in Fig. 2, the large nodes represent the locations of PoIs, among which the red ones represent the schools, and blue ones represent other PoIs. The small black nodes represent the locations of data sources (e.g., sensors). Assuming a requirement $\boxdot_{([0,+\infty],\text{School})}\Box_{[a,b]}(\mathcal{A}^{\text{op}}_{([0,d],\top)}\varphi \sim c)$ requires to aggregate and check φ only nearby schools (i.e., the red circles), but it will actually check data sources of all nearby 12 nodes if one is following the traditional parsing algorithm. In New York City, there are about 2000 primary schools, but hundreds of thousands of PoIs in total. A very large amount of computing time would be wasted in this way.

To deal with this problem, we now introduce a monitoring cost function $\mathsf{cost} : \Phi \times L \times G_L \rightarrow \mathbb{R}^+$, where Φ is the set of all the possible SaSTL formulas, L is the set of locations, G_L is the set of all the possible undirected graphs with L locations. The cost function for φ is defined as

$$\mathsf{cost}(\varphi, l, G)$$
$$= \begin{cases} 1, & \text{if } \varphi := p \vee \varphi := x \sim c \vee \\ & \quad \varphi := \text{True} \\ 1 + \mathsf{cost}(\varphi_1, l, G), & \text{if } \varphi := \neg\varphi_1 \\ \mathsf{cost}(\varphi_1, l, G) + \mathsf{cost}(\varphi_2, l, G), & \text{if } \varphi := \varphi_1 * \varphi_2, * \in \{\wedge, \mathcal{U}_I\} \\ |L^l_{\mathcal{D}}|, & \text{if } \varphi := \mathcal{A}^{\text{op}}_{\mathcal{D}} x \sim c \\ |L^l_{\mathcal{D}}|\mathsf{cost}(\varphi_1, l, G), & \text{if } \varphi := \mathcal{C}^{\text{op}}_{\mathcal{D}} \varphi_1 \sim c. \end{cases}$$

Using the above function, the cost of each operation is calculated before "switch φ" (refer to Algorithm 1). The cost function measures how complex it is to monitor a particular SaSTL formula. This can be used when the algorithm evaluates the $\wedge$ operator and it establishes the order in which

Algorithm 4: Satisfaction of $(\varphi_1 \wedge \varphi_2, \omega)$

```
case $\varphi_1 \wedge \varphi_2$ do
    return Monitor($\varphi_1, \omega, t, l, G$) $\wedge$ Monitor($\varphi_2, \omega, t, l, G$);
    if $cost(\varphi_1, l, G) \leq cost(\varphi_2, l, G)$ then
        if $\neg$ Monitor($\varphi_1, \omega, t, l, G$) then
            return Monitor($\varphi_2, \omega, t, l, G$);
        end
        return True;
    end
    if $\neg$ Monitor($\varphi_2, \omega, t, l, G$) then
        return Monitor($\varphi_1, \omega, t, l, G$);
    end
    return True;
end
```

Algorithm 5: Parallelization of Counting of $(x, op, \mathcal{D}, \omega, t, l, G)$

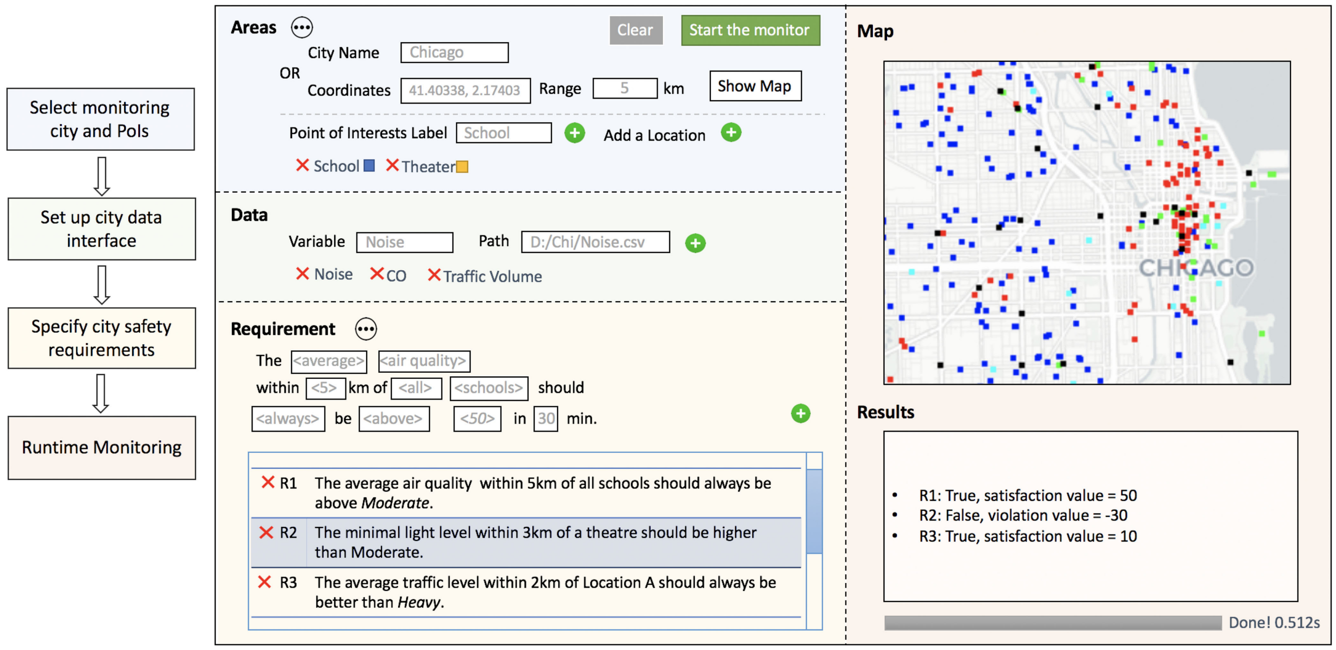
```
Function
 CountingNeighbours($\varphi, op, \mathcal{D}, \omega, t, l, G$):
    begin
        paratasks = Queue();
        for $l' \in L^l_{\mathcal{D}}$ do
            paratasks.add($l$);
        end
        results = Queue();
        for $i$ in 1..NumThreads do
            Thread$_i$ $\leftarrow$
              worker($\varphi, \omega, t, G$);
        end
        Wait();
        return op(results);
    end

Function worker ($\varphi, \omega, t, G$):
    begin
        Real $v := 0$;
        if $op$ == "min" then
          $v := \infty$; ;
        if $op$ == "max" then
          $v := -\infty$; ;
        while Num(tasks)>0 do
            $l$ = paratasks.pop();
            moni =
              Monitor($\varphi, \omega, t, l, G$);

            v = op(v, moni);
        end
        results.add(v)
    end
```

the subformulas should be evaluated. The simpler subformula is the first to be monitored, while the more complex one is monitored only when the other subformula is satisfied. We update $\mathsf{monitor}(\varphi_1 \wedge \varphi_2, \omega)$ in Algorithm 4. With this cost function, the time complexity of the monitoring algorithm is reduced to $O(|\phi| \times T_{\max} \times (\log(n) + |L'|_{\max}))$, where $|L'|$ is the maximal number of locations that an operation is executed with the improved parsing method. The improvement is significant for city requirements, where $|L'|_{\max} < 100 \times |L|_{\max}$.

C. Parallelization

In the traditional STL monitor algorithm, the signals are checked sequentially. For example, to see if the data streams from all locations satisfy $\boxdot_{\mathcal{D}}\Box_{[a,b]}\varphi$ in Fig. 2, usually, it would first check the signal from location 1 with $\Box_{[a,b]}\varphi$, then location 2, and so on. Finally, it calculates the result from all locations with $\boxdot_{\mathcal{D}}$. In this example, checking all locations sequentially is the most time-consuming part, and it could reach over 100 locations in the field.

To reduce the computing time, we parallelize the monitoring algorithm in the spatial domain. To briefly explain the idea: instead of calculating a subformula $(\Box_{[a,b]}\varphi)$ at all locations sequentially, we distribute the tasks of monitoring independent locations to different threads and check them in parallel. (Algorithm 5 presents the parallel version of the spatial counting operator $\mathcal{C}_{\mathcal{D}}$.) To start with, all satisfied locations



City Name: Chicago
Coordinates: 41.40338, 2.17403 Range: 5 km
Point of Interests Label: School
✕ School 🟦 ✕ Theater 🟨
Variable: Noise Path: D:/Chi/Noise.csv
✕ Noise ✕ CO ✕ Traffic Volume
The <average> <air quality> within <5> km of <all> <schools> should <always> be <above> <50> in 30 min.

✕ R1 The average air quality within 5km of all schools should always be above *Moderate*.
✕ R2 The minimal light level within 3km of a theatre should be higher than Moderate.
✕ R3 The average traffic level within 2km of Location A should always be better than *Heavy*.

- R1: True, satisfaction value = 50
- R2: False, violation value = -30
- R3: True, satisfaction value = 10

Fig. 3. Interface of the SaSTL monitoring tool.

$l' \in L^l_D$ are added to a task pool (a queue). In the mapping process, each thread retrieves monitoring tasks (i.e., for $l_i, \square_{[a,b]}\varphi$) from the queue and executes them in parallel. All threads only execute one task at one time and is assigned a new one from the pool when it finishes the last one, until all tasks are executed. Each task obtains the satisfaction of $\mathsf{Monitor}(\varphi, \omega, t, l, G)$ function and calculates the local result v_i of operation $\mathsf{op}()$. The reduce step sums all the parallel results and calculates a final result of $\mathsf{op}()$.

Lemma 2: The time complexity of the parallelized algorithm Monitor(ϕ, ω) is upper bounded by $O(|\phi|T_{\max}(\log(n) + [|L|_{\max}/P]))$ when distributed to P threads.

In general, the parallel monitor on the spatial domain reduces the computational time significantly. It is very helpful to support runtime monitoring and decision making, especially for a large number of requirements to be monitored in a short time. In practice, the computing time also depends on the complexity of temporal and spatial domains, as well as the amount of data to be monitored. A comprehensive experimental analysis of the time complexity is presented in Section VII.

VI. Tool for the SaSTL Monitor

We develop a user-friendly prototype tool for the SaSTL monitor that can support decision making of different stakeholders in smart cities. The interface and flowchart of the tool are shown in Fig. 3. The tool allows users (e.g., city decision maker, citizens) without any formal method background to check the city performance (data) with their own requirements easily in four steps.

Step 1 (Selecting the Monitoring City and PoI): To start with, users select the areas (such as a city or a particular area of the city) to monitor, then choose the important labels that a requirement is involved with, such as schools, parks, theaters, etc. Once selected, the important PoIs are shown on the map.

T:= The <aggregation operator> <entity> within <d> miles (from <a>th mile to <b>th mile) of <spatial operator> <PoIs> should <temporal operator> be <compare> <parameter> within <t> hours (from <m>th hour to <n>th hour / on <date> day).
T:= If **T1**, then **T2**.
T:= It is prohibited that **T1**.
T:= **T1** and/until/except **T2**.

Fig. 4. Templates to specify city requirements.

This helps users define and verify the monitoring locations. If a location or label is not included, users are also able to add them with their GPS coordinates. The map displays the locations of the specified labels and sensors. Users can enlarge the map to check the distribution of sensors and PoIs and revise the requirements accordingly.

Step 2 (Setting Up the City Data Interface): The data of the city states collected from sensors across temporal and spatial domains are introduced to the monitor in the Data section. For the offline monitoring, users can specify the data location of each variable on the computer. For runtime monitoring, the sensing data continuously come into the computer, the data interface of which can be set up in this section.

Step 3 (Specifying the City Safety Requirements): As the next important step, users specify all requirements in the requirement section. Users first select the template and then choose/fill in the essential part using the structured template language. To be noted, the entities and spatial ranges correspond to the available data variables and PoIs inputs from the areas and data sections.

We define a series of templates using structured language learning from the existing city requirements, as shown in Fig. 4. The goal of these templates is to help and inspire users to specify requirements precisely. These templates are adequate to represent all the example requirements given in

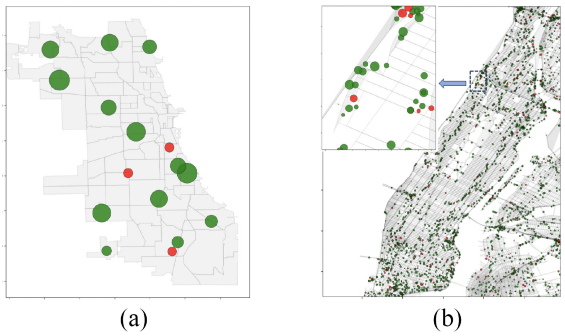


Fig. 5. Display of the monitoring results on the maps (the green circle represents the location satisfied the requirement and the red circle represents the location violates the requirement; the size of the circle represents the degree of satisfaction or violation).

Table I as well as the total set of 1000 quantitatively defined requirements. We define the templates in a recursive way. T is a template, and $T1$ and $T2$ are instances of T. The elements in T are optional, i.e., < > can be defined as blank, indicating this element is not applicable or default in this requirement. For example, an environmental requirement is written as, "The <average> <air quality> within <1> mile of all <parks> should <always> be <above> <good>." The duration is interpreted as always (default) and there is no condition element. To convert a structured requirement to SaSTL, we extract the predefined key elements and translate them to the SaSTL formula following the rules. Meanwhile, users are also able to use the advanced features to input the city requirements in the format of the SaSTL formal formulas directly.

Step 4 (Runtime Monitoring): With all the data and requirements well defined, users can start the monitor in order to check if the incoming data from the smart city satisfies the requirements. The results are displayed with a Boolean value indicating if the requirement is satisfied and a robustness value indicating how much the requirement is satisfied or violated. In addition, the map also displays the monitor results visually. Two examples are shown in Fig. 5. The first one is monitoring an air quality requirements of high schools in Chicago, and the second one is monitoring a traffic requirement in New York City. The green circle represents the location satisfied the requirement and the red circle represents the location violates the requirement; the size of the circle represents the degree of satisfaction or violation. Users can zoom in and out the map to focus on a specific area or check the overall performance as needed [see Fig. 5(b)].

In summary, we defined templates helping users to specify requirements to the SaSTL formal formulas. We believe these templates can not only help users to convert the requirement from English to formal formulas, they are also helpful for users to write the requirements much more specifically and precisely. The templates defined in this article are not sufficient to cover all the city requirements, especially the new requirements coming with more and more smart services being developed. However, the approach that using structured language to specify requirements proposed in this article is general and effective. Also, the templates are easily extended to adapt to new requirements.

We envision this tool can be used by different stakeholders in smart cities, including but not limited to the following.

City Managers and Decision Makers: In the city operating center, with city data collected in real time, the Tool is able to help city managers and decision makers to monitor the data at runtime. It also helps the city center to detect conflicts, and provide support for decision makers by showing the trade-offs of satisfaction degrees among potential solutions.

City Planners: City planners, either from the government to make long-term policies or from a company to make a short-term event plan, they are able to use the Tool to verify the past city data with their requirements and make plans to prevent the violations.

Service Designers: Smart services are designed by different stakeholders, including the government, companies, and private parties, they are not aware of all the other services. However, with the monitor, they can test the influence of their services on the city and adjust the services to better serve the city.

Everyday Citizens: The tool can also provide a service to the everyday citizens. Citizens without any technical background are able to specify their own requirements and check them with the city data to find out in which areas of the city and period of the day their requirements are satisfied, and make plans about their daily life. For example, a citizen can specify an environmental requirement with his/her preferred air quality index and traffic conditions, and check the city data with the requirements and make up traveling agenda accordingly.

VII. Evaluation

We evaluate the SaSTL monitor by applying it to three big city application scenarios: 1) *New York*; 2) *Chicago*; and 3) *Aarhus*. The experiments are evaluated on a server machine with 20 CPUs, each core is 2.2 GHz, and 4 Nvidia GeForce RTX 2080Ti GPUs. The operating system is Centos 7.

A. Runtime Monitoring of Real-Time Requirements in Chicago

1) Introduction: We apply SaSTL to monitor the real-time requirements in Chicago. The framework is the same as shown in Fig. 1, where we first formalize the city requirements to SaSTL formulas and then monitor the city states with the formalized requirements. Chicago is collecting and publishing city environment data (e.g., CO, NO, O3, and visible light) every day since January, 2017 [1]. In our evaluation, we emulate the Chicago data as they arrive in real time, i.e., assuming the city was operating with our SaSTL monitor. Specifically, we monitor data from 118 locations between January, 2017 and May, 2019. In addition, we incorporate the Chicago crime rate data published by the city of Chicago [30]. The sampling rates of sensors vary by locations and variables (e.g., CO is updated every few seconds, and the crime rate map is updated by events), so we normalize the data frequency as 1 min. Then, we specify 80 safety and performance requirements that are generated from the real requirements, and apply the SaSTL to monitor the data every 3 h continuously to identify the requirement violations.

2) Chicago Performance: Valuable information is identified from the monitor results of different periods during a day. We

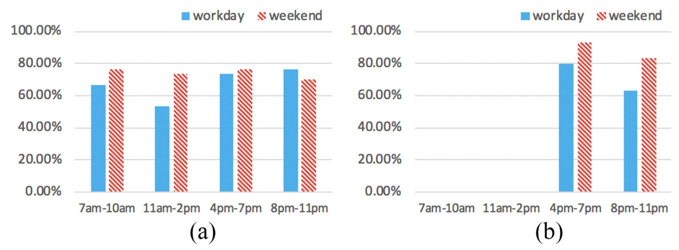


Fig. 6. Requirement satisfaction rate during different time periods in Chicago. (a) CR1. (b) CR2.

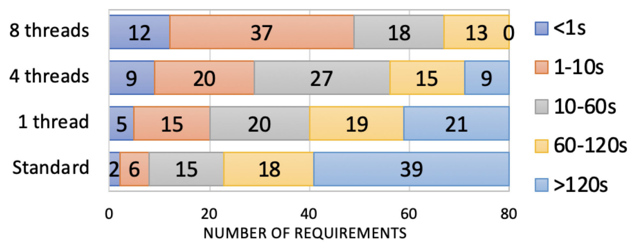


Fig. 7. Number of requirements checked on different computing time.

randomly select 30 days of weekdays and 30 days of weekends. We divide the daytime of a day into four time periods and 3 h per time period. We calculate the percentage of satisfaction (i.e., number of satisfied requirement days divides 30 days) for each time period, respectively. The results of two example requirements CR1 and CR2 are shown in Fig. 6. CR1 specifies "The average air quality within 5 km of all schools should always be above *Moderate* in the next 3 h." and is formalized as $\boxdot_{([0,+\infty),\mathsf{School})}\Box_{[0,3]}(\mathcal{A}^{\text{avg}}_{([0,5],\top)}x_{\mathsf{air}} > \mathsf{Moderate})$. CR2 specifies "for the blocks with a high crime rate, the average light level within 3 km should always be *High*" and is formalized as $\boxdot_{([0,+\infty),\top)}\Box_{[0,3]}(x_{\mathsf{Crime}} = \mathsf{High} \rightarrow \mathcal{A}^{\text{avg}}_{([0,3],\top)}x_{\mathsf{Light}} >= \mathsf{High})$.

The SaSTL monitor results can be potentially used by different stakeholders.

First, with proper requirements defined, the city decision makers are able to identify the real problems and take actions to resolve or even avoid the violations in time. For example, from the two example requirements in Fig. 6, we could see over 20% of the time the requirements are missed everyday. Based on the monitoring results of requirement CR1, decision makers can take actions to redirect the traffic near schools and parks to improve the air quality. Another example of requirement CR2, the satisfaction is much higher (up to 33% higher in CR2, 8 P.M.–11 P.M.) over weekends than workdays. There are more people and vehicles on the street on weekends, which as a result also increases the lighted areas. However, as shown in the figure, the city lighting in the areas with high crime rate is only 60%. An outcome of this result for city managers is that they should pay attention to the illumination of workdays or the areas without enough light to enhance public safety.

Second, it gives the citizens the ability to learn the city conditions and map that to their own requirements. They can make decisions on their daily living, such as the good time to visit a park. For example, requirement CR1, 11 A.M.–2 P.M. has the lowest satisfaction rate of the day. The instantaneous air quality seems to be fine during the rush hour, but it has an accumulative result that affects citizens' (especially students and elderly people) health. A potential suggestion for citizens who visit or exercise in the park is to avoid 11 A.M.–2 P.M.

3) Algorithm Performance: We count the average monitoring time taken by each requirement when monitoring for 3-h data. Then, we divide the computing time into five categories, i.e., less than 1 s, 1–10 s, 10–60 s, 60–120 s, and longer than 120 s, and count the number of requirements under each category under the conditions of standard parsing, improved parsing with single thread, four threads, and eight threads. The results are shown in Fig. 7. Comparing the 1st (standard parsing) and 4th (eight threads) bar, without the improved monitoring algorithms, for about 50% of the requirements, each one takes more than 2 min to execute. The total time of monitoring all 80 requirements is about 2 h, which means that the city decision maker can only take actions to resolve the violation at earliest 5 h later. However, with the improved monitoring algorithms, for 49 out of 80 requirements, each one of them is executed within 60 s, and each one of the rest requirements is executed within 120 s. The total execution time is reduced to 30 min, which is a reasonable time to handle as many as 80 requirements. More importantly, it illustrates the effectiveness of the parsing function and parallelization methods. Even if there are more requirements to be monitored in a real city, it is doable with our approach by increasing the number of processors.

B. Runtime Conflict Detection and Resolution in Simulated New York City

1) Introduction: The framework of runtime conflict detection and resolution [10], [31] considers a scenario, where smart services send action requests to the city center, and where a simulator predicts how the requested actions change the current city states over a finite future horizon of time. Then, it checks the predicted states against city requirements. If the requirements are satisfied, the requested actions will be approved to execute in the city. If there exists a requirement violation within the future horizon, a conflict is detected. CityResolver will be applied to resolve the conflicts. Details of the resolution are not the main part of this article, please refer to CityResolver [10]. Note that with the conflicts detected and resolved, the city's future states will be affected. In this article, we apply the SaSTL monitor to specify requirements with spatial aggregation and check the *predicted spatial–temporal data* with the SaSTL formulas.

We set up a smart city simulation of New York City using the simulation of urban mobility (SUMO) [32] with the traffic pattern (vehicle in-coming rate of key streets) from real city data [33], on top of which, we implement ten services (*S1:* Traffic Service, *S2:* Emergency Service, *S3:* Accident Service, *S4:* Infrastructure Service, *S5:* Pedestrian Service, *S6:* Air Pollution Control Service, *S7:* PM2.5/PM10 Service, *S8:* Parking Service, *S9:* Noise Control Service, and *S10:* Event Service). The real-time states (including CO, NO, O3, PMx, Noise, Traffic, Pedestrian Number, Signal Lights, Emergency Vehicles, and Accident number) from the domains of environment, transportation, events and emergencies are obtained from about 10 000 simulated nodes. Then, we apply the STL

TABLE III
SAFETY AND PERFORMANCE REQUIREMENTS FOR NEW YORK CITY

	Requirement	SaSTL
NYR1	The average noise level in the school area (within 1km) should always be less than 50dB in the next 30min.	$\boxdot_{([0,+\infty),\mathsf{School})} \square_{[0,30]} (\mathcal{A}^{\text{avg}}_{([0,1],\top)} x_{\mathsf{Noise}} < 50)$
NYR2	If an accident happens, at least one of the nearby hospitals (within 5km), its traffic condition within 2km should not reach the level of congestion in the next 60 min.	$\boxdot_{([0,+\infty),\top)}(\mathsf{Accident} \rightarrow \mathcal{C}_{([0,5],\mathsf{Hospital})}(\square_{[0,60]}(\mathcal{A}^{\text{avg}}_{([0,2],\top)} x < \mathsf{Congestion})) > 0)$
NYR3	If there is an event, the max number of pedestrians waiting at an intersection should not be greater than 50 for more than 10 minutes.	$\boxdot_{([0,+\infty),\top)}(\mathsf{Event} \rightarrow \square_{[0,10]}(\mathcal{A}^{\max}_{([0,1],\top)} x_{\mathsf{ped}} < 50))$
NYR4	At least 90% of the streets, the PMx emission should not exceed *Moderate* in 60 min.	$\mathcal{C}^{\text{avg}}_{([0,+\infty),\top)}(\square_{[0,60]}(\mathcal{A}^{\max}_{([0,1],\top)} x_{\mathsf{PMx}} < \mathsf{Moderate})) > 0.9$
NYR5	If an accident happens, it should be solved within 60 min, and before that nearby (500 m) traffic should be above moderate on average and safe in worst case.	$\boxdot_{([0,+\infty),\top)}(\mathsf{Accident} \rightarrow (\mathcal{A}^{\text{avg}}_{([0,500],\top)} x_{\mathsf{traffic}} < \mathsf{Moderate} \wedge \mathcal{A}^{\max}_{([0,500],\top)} x_{\mathsf{traffic}} < \mathsf{Safe})\mathcal{U}_{[0,60]} \neg\mathsf{Accident})$

TABLE IV
COMPARISON OF THE CITY PERFORMANCE WITH THE STL MONITOR AND THE SASTL MONITOR

	No Monitor	SaSTL Monitor
Number of Violation	Unknown	**173**
Air Quality Index	67.91	**40.18**
Noise (db)	73.32	**41.42**
Emergency Waiting Time (s)	20.32	**11.88**
Vehicle Waiting Number	22.7	**12.6**
Pedestrian Waiting Time (s)	190.2	**61.1**
Vehicle Waiting Time (s)	112.12	**59.22**

TABLE V
COMPUTING TIME OF REQUIREMENTS WITH STANDARD PARSING FUNCTION, WITH IMPROVED PARSING FUNCTIONS AND DIFFERENT NUMBER OF THREADS

	Standard Parsing (s)	1 thread (s)	4 threads (s)	8 threads (s)
NYR1	2102.13	140.29	50.31	**26.12**
NYR2	55.2	**0.837**	1.023	0.912
NYR3	69.22	22.25	7.54	**4.822**
NYR4	390.19	390.19	100.23	**53.32**
NYR5	61.76	61.76	20.25	**15.68**
Total	2678.5	615.32	179.35	**100.85**

Monitor as the *baseline* to compare the capability of requirement specification and the ability to improve city performance. We simulate the city running for 30 days with sampling rate as 10 s in two control sets, one without any monitor and one with the SaSTL monitor. For the first set (no monitor), there is no requirement monitor implemented. For the second one (SaSTL monitor), five examples of different types of real-time requirements and their formalized SaSTL formulas are given in Table III.

2) NY City Performance: The results are shown in Table IV. We measure the city performance from the domains of transportation, environment, emergency and public safety using the following metrics, the total number of violations detected (i.e., the total number of safety requirements violated during the whole simulation time), the average CO (mg) emission per street, the average noise (dB) level per street, the emergency vehicles waiting time per vehicle per intersection, the average number and waiting time of vehicles waiting in an intersection per street, and the average pedestrian waiting time per intersection.

We make some observations by comparing and analyzing the monitoring results.

First, the SaSTL monitor obtains a better city performance with fewer number of violations detected under the same scenario. As shown in Table IV, on average, the framework of conflict detection and resolution with the SaSTL monitor improves the air quality by 40.8%, and improves the pedestrian waiting time by 47.2% comparing to the one without a monitor.

Second, the SaSTL monitor reveals the real city issues, helps refine the safety requirements in real time and supports improving the design of smart services. We also compare the number of violations on each requirement. The results [Fig. 8(a)] help the city managers to measure city's performance with smart services for different aspects, and also help policymakers to see if the requirements are too strict to be satisfied by the city and make a more realistic requirement if

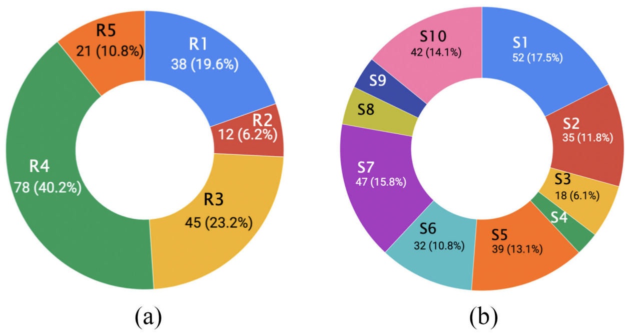


Fig. 8. Distributions of the violations over (a) requirements and (b) smart services.

necessary. For example, in our 30-day simulation, apparently, NYR4 on air pollution is the one requirement that is violated by most of the smart services. Similarly, Fig. 8(b) shows the number of violations caused by different smart services. Most of the violations are caused by $S1$, $S5$, $S6$, $S7$, and $S10$. The five major services in total cause 71.3% of the violations. City service developers can also learn from these statistics to adjust the requested actions, the inner logic and parameters of the functions of the services, so that they can design a more compatible service with more acceptable actions in the city.

3) Algorithm Performance: We compare the average computing time for each requirement under four conditions: 1) using the standard parsing algorithm without the cost function; 2) improved parsing algorithm with a single thread; 3) improved parsing algorithm with spatial parallelization using four threads; and 4) using eight threads. The results are shown in Table V.

First, the improved parsing algorithm reduces the computing time significantly for the requirement specified on PoIs, especially for NYR1 that computing time reduces from 2102.13 s to 140.29 s (about 15 times). Second, the parallelization over spatial operator further reduces the computing time in most of the cases. For example, for NYR1, the computing time is

TABLE VI
SAFETY AND PERFORMANCE REQUIREMENTS FOR AARHUS

	Requirement	**SaSTL**
AR1	If there is an event, the traffic level nearby should always be better than *Moderate*.	Event $\rightarrow \boxdot_{\mathcal{D}} \square_{[0,3]} x_{\text{traffic}} >$ Moderate
AR2	If there is an event, the average traffic level nearby should always be better than *Moderate* and the maximum traffic level nearby should be better than *Safe*.	Event $\rightarrow \boxdot_{\mathcal{D}} \square_{[0,3]} (\mathcal{A}^{\text{avg}}_{([0,1],\top)} x_{\text{traffic}} >$ Moderate $\wedge\, \mathcal{A}^{\max}_{([0,1],\top)} x_{\text{traffic}} >$ Safe)
AR3	If there is an event, the average traffic near the school (3km) should always be better than *Moderate* and the maximum traffic level should be better than *Heavy*.	Event $\rightarrow \boxdot_{\mathcal{D}} \square_{[0,3]} (\mathcal{A}^{\text{avg}}_{([0,1],\top)} x_{\text{traffic}} >$ Moderate $\wedge\, \mathcal{A}^{\max}_{([0,1],\top)} x_{\text{traffic}} >$ Heavy) $\wedge$ School
AR4	If there is an event and the weather is rainy or snowy heavily, the average traffic level around school should be better than *Heavy*	Event $\wedge$ Humidity $> 50\% \rightarrow \boxdot_{\mathcal{D}} \square_{[0,3]} (\mathcal{A}^{\text{avg}}_{([0,1],\top)} x_{\text{traffic}} >$ Heavy) $\wedge$ School
AR5	With big cultural events going on the city, over the city, 80% schools' average traffic volume nearby (3km) should always be better than *Moderate*.	Event $\rightarrow \mathcal{C}_{\mathcal{D}} \square_{[0,3]} (\mathcal{A}^{\text{avg}}_{([0,1],\top)} x_{\text{traffic}} >$ Moderate) $\wedge$ School $> 80\%$

reduced to 26.12 s with eight threads while 140.29 s with single thread (about five times). When the amount of data is very small (NYR2), the parallelization time is similar to the single thread time, but still much efficient than the standard parsing.

The results demonstrate the effectiveness and importance of the efficient monitoring algorithms. In the table, the total time of monitoring five requirements is reduced from 2678.5 to 100.85 s. In the real world, when multiple requirements are monitored simultaneously, the improvement is extremely important for real-time monitoring.

C. Evaluation for Aarhus

1) Introduction: In this case study, we monitor the past data of events and states from Aarhus to show how the SaSTL monitor helps to understand the effects caused by events and, therefore, aids in decision making for city events. We utilize 60 days (August 2014 to September 2014) of Aarhus city data collected simultaneously across the domains of transportation (e.g., traffic volume and parking), events (e.g., cultural events and library events) and the environment (generated pollution and weather). All the data were collected from 449 observation points and published by CityPulse [34]. Data were collected with different sampling rates (e.g., the traffic data were aggregated by 5 min and events data were recorded by the event time), thus for the monitoring purpose, we normalize the data frequency as 5 min. Five safety and performance requirements and their corresponding SaSTL statements are presented with a high demand for aggregations specified for Aarhus in Table VI. Basically, AR1–AR5 specify that when there is an event, there is a different level of safety requirements on the traffic under different circumstances. For example, AR2 focuses on the areas nearby an event, AR3 focuses on the safety of school with an event, and R4 considers the effects from extreme weather conditions. AR5 has a big picture on all schools across the city when a large cultural event is happening.

2) Performance: The monitoring results from Aarhus are shown in Fig. 9. The percentage of satisfaction equals the number of requirement satisfied days divided by 60 days. The following are observations on the requirements and monitoring results.

1) Comparing the monitoring results on AR1 and AR2, AR1 has a much lower satisfaction rate. It also leads to a higher and reliable satisfaction rate.
2) Compared to AR2, for the same events, AR3 moves its focus on the area nearby schools. The results, however, are lower than AR2. It means that events have more influence on the school areas, which should draw attention from the city managers. Students should reduce or avoid activities during this time when there is an event going on nearby.
3) During 11 A.M. to 2 P.M., the overall performance on all five requirements is worst, even less than 50%. It is actually the time period right after a morning event or before an afternoon event. The monitoring results help the city managers have a better view of the distribution of effects from events.
4) We also find that the satisfaction rate is very high (almost 100%) after 8 P.M. The reasons for that are the schools are usually closed at that time, and most of the cultural and library events happen during the day. In other cities or events, the distribution will be different. However, the SaSTL monitor is general enough to help citizens and managers detect it.

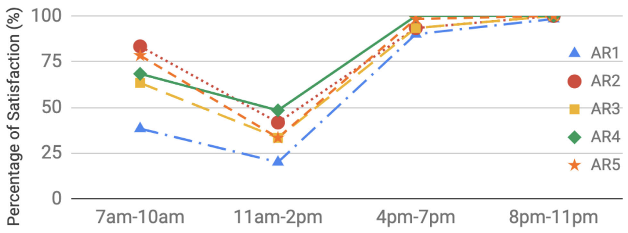


Fig. 9. Comparisons of satisfaction rate on AR1–AR5.

The evaluation on Aarhus shows how the SaSTL monitor helps the city to understand the effects on the city from events and make better plans for events. Usually, areas with an event get caught up in complicated situations, such as paralyzed traffic, long queues with a large amount of people, emergencies, and accidents. Therefore, playing back and analyzing the city data during events is extremely important for cities to avoid emergency situations for future events.

VIII. COVERAGE ANALYSIS

We compare the specification coverage on 1000 quantitatively specified real city requirements between STL, SSTL, STREL, and SaSTL. The study is conducted by graduate students following the rules that if the language is able to specify the whole requirement directly with one single formula, then it is identified as True. To be noted, another spatial STL, SpaTeL, is not considered as a baseline here, because it is not applicable to most of city spatial requirements. SpaTeL is built on a quad tree, and able to specify directions rather than the distance.

STL is only able to specify 184 out of 1000 requirements, while SSTL and STREL are able to formalize 431 requirements. SaSTL is able to specify 950 out of 1000 requirements.

In particular, we made the following observations from the results. First, 50 requirements cannot be specified using any of the four languages because they are defined by complex math formulas that are ambiguous with missing key elements, relevant to the operations of many variables, or referring to a set of other requirements, e.g., "follow all the requirements from Section 201.12," etc. Second, SSTL, STREL, and SaSTL outperformed STL in terms of requirements with spatial ranges, such as "one-mile radius around the entire facility"; third, SSTL and STREL have the same coverage on the requirements that only contain a temporal and spatial range. Comparing to SSTL and SaSTL, STREL can also be applied to the dynamic graph and check requirements reachability, which is very useful in applications like wireless sensor networks, but not common in smart city requirements; fourth, the rest of the requirements (467 out of 1000) measure the aggregation of a set of locations, which can only be specified using SaSTL.

IX. Related Work

Monitoring spatial–temporal properties over CPS executions has been initially investigated in [35] and [36], where the authors introduced a spatial–temporal event-based model for monitoring CPS. In this model, events are labeled with time and space stamps. These events can be triggered by actions, exchange of messages, or physical changes. A centralized monitor is then responsible to process all these events. Their approach provides an algorithmic framework enabling a user to develop manually a monitor, but they do not provide any spatial–temporal specification language. The literature instead offers several logic-based specification languages to reason about the spatial structure of the concurrent systems [37], medical images [38], and the topological [39] or directional [40] aspects of the interacting components. However, these logics are not practical for monitoring CPS, because they are generally computationally complex [40] or even undecidable [41].

Specification-based monitoring of spatial–temporal properties over CPS executions has become practical only recently with SpaTeL [9] and SSTL [12]. SpaTeL extends the STL [11] with the tree spatial superposition logic (TSSL) [42], [43]. TSSL classifies and detects spatial patterns by reasoning over-quad trees, suitable spatial data structures that are constructed by recursively partitioning the space into uniform quadrants. The notion of superposition in TSSL [43] provides a way to describe statistically the distribution of discrete states in a particular partition of the space and the spatial operators corresponding to *zooming in and out in* a particular region of the space. By nesting these operators, it is possible to specify self-similar and fractal-like structures [44] that generally characterize the patterns emerging in nature such as the electrical spiral formation in cardiac tissues [45]. The procedure allows one to capture very complex spatial structures, but at the price of a complex formulation of spatial properties, which are in practice only learned from some template image.

SSTL [12] extends STL with several spatial operators (i.e., somewhere, everywhere, and surround). The SSTL semantics operates on a weighted undirected graph, where the weight on each edge represents the distance between two nodes. The spatial–temporal reach and escape logic (STREL) [13], [14] generalizes SSTL, by introducing two new spatial operators: 1) *reach* and 2) *escape*, which are able to express the same spatial operators of SSTL. Furthermore, while SSTL can be applied only on static weight undirected graphs, STREL can be applied also to dynamic networks. However, both SSTL and STREL do not support spatial aggregation operators that we show to be an important feature for monitoring smart cities.

X. Conclusion

In this article, we presented a novel SaSTL to specify and to monitor requirements of smart cities at runtime. We develop an efficient monitoring framework that optimizes the requirement parsing process and can check in parallel an SaSTL requirement over multiple data streams generated from thousands of sensors that are typically spatially distributed over a smart city. SaSTL is a powerful specification language for smart cities because of its capability to monitor the city desirable features of temporal (e.g., interval), spatial (e.g., PoIs and range), and their complicated relations (e.g., always, everywhere, and aggregation) between them. More importantly, it can coalesce many requirements into a single SaSTL formula and provide the aggregated results efficiently, which is a major advance on what smart cities do now. The development of 5G and 6G could better support the monitoring and communication among sensors, services and the city center. We believe it is a valuable step toward developing a practical smart city monitoring system even though there are still open issues for future work. Furthermore, SaSTL monitor can also be easily generalized and applied to monitor other large-scale IoT deployments at runtime efficiently. In the future, we will explore its capability to specify and monitor other properties and requirements (e.g., security and privacy).

References

[1] C. E. Catlett, P. H. Beckman, R. Sankaran, and K. K. Galvin, "Array of things: A scientific research instrument in the public way: Platform design and early lessons learned," in *Proc. ACM 2nd Int. Workshop Sci. Smart City Oper. Platforms Eng.*, 2017, pp. 26–33.

[2] *IBM Takes 'Smarter Cities' to Rio De Janeiro*, New York Times, New York, NY, USA, 2012.

[3] *Smart+Connected Operations Center*, Cisco, San Jose, CA, USA, 2017.

[4] M. Ma, S. M. Preum, M. Y. Ahmed, W. Tärneberg, A. Hendawi, and J. A. Stankovic, "Data sets, modeling, and decision making in smart cities: A survey," *ACM Trans. Cyber Phys. Syst.*, vol. 4, no. 2, pp. 1–28, 2019.

[5] M. Ma, S. M. Preum, and J. A. Stankovic, "CityGuard: A watchdog for safety-aware conflict detection in smart cities," in *Proc. 2nd Int. Conf. Internet Things Design Implement.*, 2017, pp. 259–270.

[6] H. Zhang, Y. Zheng, and Y. Yu, "Detecting urban anomalies using multiple spatio-temporal data sources," *ACM Interact. Mobile Wearable Ubiquitous Technol.*, vol. 2, no. 1, p. 54, 2018.

[7] S. Sheng *et al.*, "A case study of trust on autonomous driving," in *Proc. IEEE Intell. Transp. Syst. Conf. (ITSC)*, 2019, pp. 4368–4373.

[8] M. Ma, J. A. Stankovic, and L. Feng, "Runtime monitoring of safety and performance requirements in smart cities," in *Proc. 1st ACM Workshop Internet Safe Things*, 2017, pp. 44–50.

[9] I. Haghighi, A. Jones, Z. Kong, E. Bartocci, R. Gros, and C. Belta, "SpaTel: A novel spatial–temporal logic and its applications to networked systems," in *Proc. ACM 18th Int. Conf. Hybrid Syst. Comput. Control*, 2015, pp. 189–198.

[10] M. Ma, J. A. Stankovic, and L. Feng, "Cityresolver: A decision support system for conflict resolution in smart cities," in *Proc. 9th ACM/IEEE Int. Conf. Cyber Phys. Syst.*, 2018, pp. 55–64.

[11] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Proc. FORMATS*, 2004, pp. 152–166.

[12] L. Nenzi, L. Bortolussi, V. Ciancia, M. Loreti, and M. Massink, "Qualitative and quantitative monitoring of spatio-temporal properties," in *Proc. 6th Int. Conf. Runtime Verification (RV)*, vol. 9333, 2015, pp. 21–37.

[13] E. Bartocci, L. Bortolussi, M. Loreti, and L. Nenzi, "Monitoring mobile and spatially distributed cyber-physical systems," in *Proc. 15th ACM/IEEE Int. Conf. Formal Methods Models Syst. Design (MEMOCODE)*, 2017, pp. 146–155.

[14] E. Bartocci, L. Bortolussi, M. Loreti, L. Nenzi, and S. Silvetti, "MoonLight: A lightweight tool for monitoring spatio-temporal properties," in *Proc. 20th Int. Conf. Runtime Verification (RV)*, vol. 12399, 2020, pp. 417–428.

[15] L. Nenzi, E. Bartocci, L. Bortolussi, M. Loreti, and E. Visconti, "Monitoring spatio-temporal properties (invited tutorial)," in *Proc. 20th Int. Conf. Runtime Verification (RV)*, vol. 12399, 2020, pp. 21–46.

[16] M. Ma, E. Bartocci, E. Lifland, J. Stankovic, and L. Feng, "SaSTL: Spatial aggregation signal temporal logic for runtime monitoring in smart cities," in *Proc. ACM/IEEE 11th Int. Conf. Cyber Phys. Syst. (ICCPS)*, 2020, pp. 51–62.

[17] NYC.gov. (2019). *Emissions From Transportation, Nyc Environment Protection*. [Online]. Available: https://www1.nyc.gov/html/dep/html/air/emissions_from_transportation.shtml

[18] *Air Quality—Motor Vehicular Pollutants, Lead, Odors, and Nuisance Pollutants*, District Columbia Municipal Regul., Washington, DC, USA, 2016.

[19] S. Matteo and J. Brannan, *A Local Law to Amend the Administrative Code of the City of New York, in Relation to Restricting the Use of Bus Lanes by Sight-Seeing Buses*, New York City Council, New York, NY, USA, 2019.

[20] NYC Environment Protection, *Use of Heating Oil Remaining in Tanks*, City of New York, New York, NY, USA, 2019.

[21] United States Environmental Protection Agency, "Residential energy efficiency," in *Energy Resources for State and Local Governments*. New York, NY, USA: City of New York, 2019.

[22] LA Sec 111.03. Minimum Ambient Noise Level, "Official city of Los Angeles municipal code," 2016. [Online]. Available: https://codelibrary.amlegal.com/codes/los_angeles/latest/lamc/0-0-0-193850

[23] *Guide to Indoor Air Quality Management in Hong Kong Regional Offices and Public Places*, Indoor Air Qual. Manag., Hong Kong, 2019.

[24] *Stopping, Standing or Parking Prohibited in Specified Places*, New York Public Law, New York, NY, USA, 2016.

[25] *Pre-Hospital Medical Emergency Regulations*, Beijing Emerg. Agency, Beijing, China, 2016.

[26] *Safety Management for Kindergarten, Primary and Secondary School*, Beijing Govt., Beijing, China, 2016.

[27] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," *J. ACM*, vol. 27, no. 4, pp. 831–838, 1980.

[28] A. Donzé, T. Ferrere, and O. Maler, "Efficient robust monitoring for STL," in *Proc. Int. Conf. Comput.-Aided Verification*, 2013, pp. 264–279.

[29] G. S. Lueker, "A data structure for orthogonal range queries," in *Proc. 19th Annu. Symp. Found. Comput. Sci.*, 1978, pp. 28–34.

[30] City of Chicago. (2018). *Crimes of Chicago—One Year Prior to Present*. [Online]. Available: https://data.cityofchicago.org/Public-Safety/Crimes-Map/dfnk-7re6

[31] M. Ma, S. M. Preum, W. Tarneberg, M. Ahmed, M. Ruiters, and J. Stankovic, "Detection of runtime conflicts among services in smart cities," in *Proc. IEEE Int. Conf. Smart Comput. (SMARTCOMP)*, 2016, pp. 1–10.

[32] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "SUMO—Simulation of urban mobility: An overview," in *Proc. SIMUL*, 2011, pp. 31–33.

[33] *New York City Open Data*. Accessed: Mar. 1, 2021. [Online]. Available: https://nycopendata.socrata.com/

[34] C.-S. Nechifor, A. Sheth, A. Mileo, S. Bischof, A. Karapantelakis, and P. Barnaghi, "Semantic modelling of smart city data," in *Proc. W3C Workshop Web Things Enablers Services Open Web Devices* Berlin, Germany, Jun. 2014.

[35] C. L. Talcott, "Cyber-physical systems and events," in *Software-Intensive Systems and New Computing Paradigms—Challenges and Visions* (LNCS 5380). Heidelberg, Germany: Springer, 2008, pp. 101–115.

[36] Y. Tan, M. C. Vuran, and S. Goddard, "Spatio-temporal event model for cyber-physical systems," in *Proc. 29th IEEE Int. Conf. Distrib. Comput. Syst. Workshops*, 2009, pp. 44–50.

[37] L. Caires and L. Cardelli, "A spatial logic for concurrency (part I)," *Inf. Comput.*, vol. 186, no. 2, pp. 194–235, 2003.

[38] F. B. Buonamici, G. Belmonte, V. Ciancia, D. Latella, and M. Massink, "Spatial logics and model checking for medical imaging," *Int. J. Softw. Tools Technol. Transf.*, vol. 22, no. 2, pp. 195–217, 2020.

[39] B. Bennett, A. G. Cohn, F. Wolter, and M. Zakharyaschev, "Multi-dimensional modal logic as a framework for spatio-temporal reasoning," *Appl. Intell.*, vol. 17, no. 3, pp. 239–251, Sep. 2002.

[40] D. Bresolin, P. Sala, D. D. Monica, A. Montanari, and G. Sciavicco, "A decidable spatial generalization of metric interval temporal logic," in *Proc. 17th Int. Symp. Temporal Represent. Reason.*, 2010, pp. 95–102.

[41] M. Marx and M. Reynolds, "Undecidability of compass logic," *J. Logic Comput.*, vol. 9, no. 6, pp. 897–914, 1999.

[42] E. A. Gol, E. Bartocci, and C. Belta, "A formal methods approach to pattern synthesis in reaction diffusion systems," in *Proc. CDC*, 2014, pp. 108–113.

[43] E. Bartocci, E. A. Gol, I. Haghighi, and C. Belta, "A formal methods approach to pattern recognition and synthesis in reaction diffusion networks," *IEEE Trans. Control. Netw. Syst.*, vol. 5, no. 1, pp. 308–320, Mar. 2018.

[44] R. Grosu, S. A. Smolka, F. Corradini, A. Wasilewska, E. Entcheva, and E. Bartocci, "Learning and detecting emergent behavior in networks of cardiac myocytes," *Commun. ACM*, vol. 52, no. 3, pp. 97–105, 2009. [Online]. Available: http://doi.acm.org/10.1145/1467247.1467271

[45] E. Bartocci, F. Corradini, M. R. D. Berardini, E. Entcheva, S. A. Smolka, and R. Grosu, "Modeling and simulation of cardiac tissue using hybrid I/O automata," *Theor. Comput. Sci.*, vol. 410, no. 33-34, pp. 3149–3165, 2009.

Meiyi Ma is currently pursuing the Ph.D. degree in computer science with the University of Virginia, Charlottesville, VA, USA.

Her research interests are at the intersection of cyber-physical systems, deep learning, and formal methods.

Ezio Bartocci received the B.S. degree in computer science, M.S. degree in bioinformatics, and the Ph.D. in complex systems and information science from the University of Camerino, Camerino, Italy, in 2002, 2005, and 2009, respectively, and the Habilitation (Venia Docendi) degree in computer science from the Faculty of Informatics, TU Wien, Vienna, Austria, in 2019.

He is a Full Professor with the Faculty of Computer Science, TU Wien, Vienna, Austria, where he leads the Trustworthy Cyber-Physical Systems Group. His research is to develop formal methods, computational tools, and techniques that support the modeling and the automated analysis of complex computational systems, including software systems, cyber–physical systems, and biological systems.

Eli Lifland received the bachelor's degree in computer science and economics from the University of Virginia, Charlottesville, VA, USA, in 2020.

He is a Software Engineer with Ought, Washington, DC, USA.

John A. Stankovic (Life Fellow, IEEE) received the Ph.D. degree from Brown University, Providence, RI, USA, in 1979.

He is the BP America Professor with the Computer Science Department, University of Virginia, Charlottesville, VA, USA, where he is the Director of the Link Lab. His research interests are in smart and connected health, cyber–physical systems, and Internet of Things.

Dr. Stankovic has been awarded an Honorary Doctorate from the University of York for his work on real-time systems. He is a Fellow of ACM.

Lu Feng (Member, IEEE) received the Ph.D. degree in computer science from the University of Oxford, Oxford, U.K., in 2014.

She is an Assistant Professor of Computer Science with the University of Virginia, Charlottesville, VA, USA. Her research interests are in cyber–physical systems and formal methods.