SaSTL: Spatial Aggregation Signal Temporal Logic for Runtime Monitoring in Smart Cities

Meiyi Ma
Department of Computer Science
University of Virginia
Charlottesville, USA
Email: meiyi@virginia.edu

Ezio Bartocci
Faculty of Informatics
TU Wien
Vienna, Austria
Email: ezio.bartocci@tuwien.ac.at

Eli Lifland, John Stankovic, Lu Feng
Department of Computer Science
University of Virginia
Charlottesville, USA
Email: {edl9cy, stankovic, lu.feng}@virginia.edu

***Abstract*—We present SaSTL—a novel Spatial Aggregation Signal Temporal Logic—for the efficient runtime monitoring of safety and performance requirements in smart cities. We first describe a study of over 1,000 smart city requirements, some of which can not be specified using existing logic such as Signal Temporal Logic (STL) and its variants. To tackle this limitation, 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 also develop efficient monitoring algorithms that can check a SaSTL requirement in parallel over multiple data streams (e.g., generated by multiple sensors distributed spatially in a city). We evaluate our SaSTL monitor by applying to two case studies with large-scale real city sensing data (e.g., up to 10,000 sensors in one requirement). 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 can help improve the safety and performance of smart cities via simulated experiments.**

***Keywords*-Spatial Temporal Logic, runtime monitoring, requirement specification, 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], [5]. There is a need for monitoring city states in real-time to ensure safety and performance requirements [6]. 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 polices, 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 about thousands of sensors in a smart city).

Previous works [7], [8], [9], [10] have proposed solutions to monitor smart cities using formal specification languages and their monitoring machinery. One of the latest works, CityResolver [11] uses Signal Temporal Logic (STL) [12] 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 [13], SpaTeL [10] and STREL [14]), 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 Signal Temporal Logic (SaSTL), which extend STL with two new logical operators for expressing spatial aggregation and spatial counting characteristics that are commonly found in real city requirements. More specifically, this paper has the following major **contributions**:

(1) To the best of our knowledge, this is the first work studying and annotating over 1,000 real smart city requirements across different 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 requirement 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 Point of Interests (PoIs), the physical distance, spatial relations of the PoIs and sensors, aggregation of signals over locations, degree/percentage of satisfaction and the temporal elements

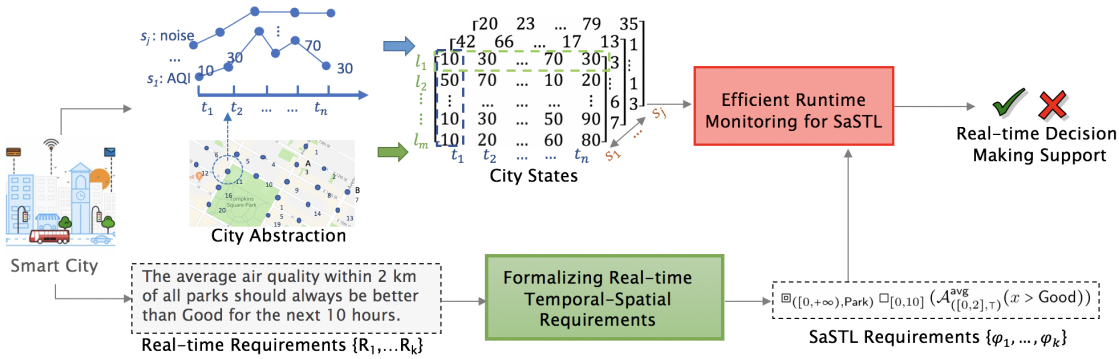


Figure 1: A framework for runtime monitoring of real-time city requirements

in a very flexible spatial-temporal scale.

(3) We compare SaSTL with some existing specification languages and show that SaSTL has a much higher coverage experessiveness (95%) than STL (18.4%), SSTL (43.1%) or STREL (43.1%) over 1,000 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: (i) dynamically prioritizing the monitoring based on cost functions assigned to nodes of the syntax tree, and (ii) parallelizing the monitoring of spatial operators among multiple locations and/or sensors.

(5) We evaluate the SaSTL monitor by applying it to a case study of monitoring real city data collected from the Chicago's Array of Things [1]. The results show that SaSTL monitor has the potential to help identify safety violations and support the city managers and citizens to make decisions.

(6) We also evaluate the SaSTL monitor on a second 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 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.

II. Approach Overview

Figure 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, noise level) over temporal and spatial domains can be viewed as a 3-dimensional 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 will 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 large cities (e.g. New York City, San Francisco, Chicago, Washington D.C., Beijing, etc.). Some examples of these city requirements are highlighted in Table I. We identify 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 and resolve requirement violations at runtime. It indicates a high demand for runtime monitoring.

Spatial: A requirement usually specifies its spatial range explicitly using the Points of Interest (PoIs) (80.1%), such as "park", "xx school", along with a distance range (65%).

Table I: Examples of city requirements from different domains (The key elements of a requirement are highlighted as, temporal, spatial, aggregation, entity, condition, 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 [15].
	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 [16].
	Prohibit sight-seeing buses from using all bus lanes between the hours of 7:00 a.m. and 10:00 a.m. on weekdays [17].
Energy	Operate the system to maintain zone temperatures down to 55°F or up to 85°F [18].
	The total leakage shall be less than or equal to 4 cubic feet per minute per 100 square feet of conditioned floor area [19].
Environment	LA Sec 111.03 minimum ambient noise level table: ZONE M2 and M3 – DAY : 65 dB(A) NIGHT : 65 dB(A) [20].
	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 [21].
Emergency	NYC Authorized emergency vehicles may disregard 4 primary rules regarding traffic [22].
	At least one ambulance should be equipped per 30,000 population (counted by area) to obtain the shortest radius and fastest response time [23].
Public Safety	Security staff shall visit at least once per week in public schools [24].

Table II: Key elements of city requirements and statistical results from 1000 real city requirements

Element	Form	Num	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...

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

IV. Formalizing Temporal-Spatial Requirements

SaSTL extends STL with two spatial operators: a *spatial aggregation* operator and 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 satisfy a certain requirement. In this section, we formally define the syntax, and semantics.

A. SaSTL Syntax

We define a multi-dimensional *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 $X = \{x_1, \cdots, x_n\}$ 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 $P = \{p_1, \cdots, p_m\}$ a set of propositions (e.g. $\{\mathsf{School}, \mathsf{Street}, \mathsf{Hospital}, \cdots\}$) and $\mathcal{L}$ a labeling function $\mathcal{L} : L \to 2^P$ 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 non-empty 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

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

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

$$\begin{aligned}\mathcal{D} &:= \quad ([d_1, d_2], \psi)\\ \psi &:= \quad \top \mid p \mid \neg\, \psi \mid \psi \ \vee \ \psi\end{aligned}$$

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. In particular, $\mathcal{D} = ([0, +\infty), \top)$ indicates the whole spatial domain. We denote $L^l_{([d_1,d_2],\psi)} := \{l' \in L \mid 0 \leq d_1 \leq d(l, l') \leq d_2$ and $\mathcal{L}(l') \vDash \psi\}$ 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 non-null values for signal variable x at time point t location l over locations in $L^l_{\mathcal{D}}$ by

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

We define a set of operations $\mathrm{op}(\alpha^x_{\mathcal{D}}(\omega, t, l))$ for $\mathrm{op} \in \{\max, \min, \mathrm{sum}, \mathrm{avg}\}$ when $\alpha^x_{\mathcal{D}}(\omega, t, l) \neq \varnothing$ 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

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

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, $\mathcal{U}_I$ 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 $\Box\varphi \equiv \neg\Diamond\neg\varphi$.

We define a set of spatial *aggregation* operators $\mathcal{A}^{\mathrm{op}}_{\mathcal{D}} x \sim c$ for $\mathrm{op} \in \{\max, \min, \mathrm{sum}, \mathrm{avg}\}$ that evaluate the aggregated product of traces $\mathrm{op}(\alpha^x_{\mathcal{D}}(\omega, t, l))$ over a set of locations $l \in L^l_{\mathcal{D}}$. We also define a set of new spatial *counting* operators $\mathcal{C}^{\mathrm{op}}_{\mathcal{D}} \varphi \sim c$ for $\mathrm{op} \in \{\max, \min, \mathrm{sum}, \mathrm{avg}\}$ that counts the satisfaction of traces over a set of locations. More precisely, we define $\mathcal{C}^{\mathrm{op}}_{\mathcal{D}} \varphi = \mathrm{op}(\{g((\omega, t, l') \vDash \varphi) \mid l' \in L^l_{\mathcal{D}}\})$, where $g((\omega, t, l) \vDash \varphi)) = 1$ if $(\omega, t, l) \vDash \varphi$, otherwise $g((\omega, t, l) \vDash \varphi)) = 0$. 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, $\mathcal{C}^{\mathrm{sum}}_{\mathcal{D}} \varphi$ specifies the total number of locations that satisfy φ and $\mathcal{C}^{\mathrm{avg}}_{\mathcal{D}} \varphi$ 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 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 hours" is formalized as,* $\boxdot_{([0,+\infty),\mathsf{School})} \Box_{[0,3]} ((\mathcal{A}^{\mathrm{avg}}_{([0,1],\top)} x_{\mathsf{Noise}} < 50) \wedge (\mathcal{A}^{\mathsf{max}}_{([0,1],\top)} x_{\mathsf{Noise}} < 80))$.

$([0, +\infty), \mathsf{School})$ 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. $(\mathcal{A}^{\mathrm{avg}}_{([0,1],\top)} x_{\mathsf{Noise}} < 50) \wedge (\mathcal{A}^{\mathsf{max}}_{([0,1],\top)} x_{\mathsf{Noise}} < 80)$ 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 hours" is formalized as* $\mathcal{C}^{\mathrm{avg}}_{([0,+\infty),\mathsf{Street})}(\Box_{[0,2]}(x_{\mathsf{PMx}} < \mathsf{Moderate})) > 0.9$.

$\mathcal{C}^{\mathrm{avg}}_{([0,+\infty),\mathsf{Street})} \varphi > 0.9$ 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* $(\omega, t, l) \vDash \varphi$, indicating that the spatio-temporal signal ω satisfies a formula φ at the time point t in location l when $\pi_v(\omega)[t, l] \neq \perp$ and $\alpha^x_{\mathcal{D}}(\omega, t, l) \neq \varnothing$. We define that $(\omega, t, l) \vDash \varphi$ if $\pi_v(\omega)[t, l] = \perp$.

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

V. Efficient Monitoring for SaSTL

In this section, we first present 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 the time t and location l), a weighted undirected graph G and the temporal-spatial data ω. The *output* of the monitoring algorithm for each requirement is a Boolean value indicating whether the requirement is satisfied or not. Algorithm 1 outlines the monitoring algorithm. To start with, the monitoring algorithm parses φ to sub formulas and calculates the satisfaction for each operation recursively. We derived operators $\Box$ and $\Diamond$ from $\mathcal{U}_I$, and operators $\boxdot$ and $\diamondsuit$ from $\mathcal{C}_{\mathcal{D}}^{\mathsf{op}} \sim c$, so we only show the algorithms for $\mathcal{U}_I$ and $\mathcal{C}_{\mathcal{D}}^{\mathsf{op}} \sim c$.

Algorithm 1: SaSTL monitoring algorithm Monitor(φ, ω, t, l, G)

```
Function Monitor(φ, ω, t, l, G):
    Input : SaSTL Requirement φ, Signal ω, Time t, Location l, weighted
            undirected graph G
    Output: Satisfaction Value (Boolean value)
    begin
        switch φ do
            Case x ∼ c
                return π_x(ω)[t, l] ∼ c;
            Case ¬φ
                return ¬ Monitor(φ, ω, t, l, G);
            Case φ_1 ∧ φ_2 ;                          ▷ See Algorithm 4
                return Monitor(φ_1, ω, t, l, G) ∧ Monitor(φ_2, ω, t, l, G)
            Case φ_1 U_I φ_2
                return SatisfyUntil(φ_1, φ_2, I, ω, t, l, G);
            Case A_D^op x ∼ c ;                       ▷ See Algorithm 2
                return Aggregate(x, c, op, D, t, l, G);
            Case C_D^op φ ∼ c ;  ▷ See Algorithm 3 for the standard version, and
              Algorithm 5 for an improved parallel version.
                return CountingNeighbours(φ, c, op, D, t, l, G);
        end
    end
```

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

```
Function Aggregate(x, c, op, D, ω, t, l, G):
    begin
        Real v := 0; n := 0;
        if op == "min" then  v := ∞ ;
        if op == "max" then  v := −∞ ;
        for l' ∈ L_D^l do
            if op ∈ {min, max, sum} then
                v := op(v, π_x(ω)[t, l']);
            end
            if op =="avg" then
                v := sum(v, π_x(ω)[t, l']);
            end
            n := n + 1
        end
        if op == "avg" ∧n ≠ 0 then  v := v/n ;
        if n == 0 then
            return True
        else
            return v ∼ c;
        end
    end
```

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

```
Function CountingNeighbours(φ, c, op, D, ω, t, l, G):
    begin
        Real v := 0; n := 0
        if op == "min" then  v := ∞ ;
        if op == "max" then  v := −∞ ;
        for l' ∈ L_D^l do
            if Monitor(φ, ω, t, l, G) ∧ op ∈ {min, max, sum} then
                v := op(v, 1);
            end
            if Monitor(φ, ω, t, l, G) ∧ op =="avg" then
                v := sum(v, 1);
            end
            n := n + 1
        end
        if op == "avg" ∧n ≠ 0 then  v := v/n ;
        if n == 0 then
            return True
        else
            return v ∼ c;
        end
    end
```

We present the satisfaction algorithms of the operators $\mathcal{A}_{\mathcal{D}}^{\mathsf{op}}$ and $\mathcal{C}_{\mathcal{D}}^{\mathsf{op}}$ in Algorithm 2 and Algorithm 3, respectively. As we can tell from the algorithms, essentially, $\mathcal{A}_{\mathcal{D}}^{\mathsf{op}}$ calculates the aggregated values on the signal over a spatial domain, while $\mathcal{C}_{\mathcal{D}}^{\mathsf{op}}$ calculates the aggregated Boolean results over spatial domain.

The time complexity of monitoring the logical and temporal operators of SaSTL is the same as STL [25] as follows.

- 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 $\Box_I$, $\Diamond_I$, $\mathcal{U}_I$ is $O(T)$, where T is the total number of samples within time interval I.

In this paper, we present the time complexity analysis for the spatial operators (Lemma 1) and the new SaSTL monitoring algorithm (Theorem 1). 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 [26] (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 (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 $\boxdot_{\mathcal{D}}$, $\diamondsuit_{\mathcal{D}}$, $\mathcal{A}_{\mathcal{D}}^{\mathsf{op}}$, and $\mathcal{C}_{\mathcal{D}}^{\mathsf{op}}$ is $O(log(n) + |L|)$ where L is the set of locations at distance within the range $\mathcal{D}$ from l.*

Proof: According to [26], the time complexity to retrieve a set of nodes L with a distance to a desired location in a range $\mathcal{D}$ from a location l is $O(log(n) + |L|)$. The aggregation and counting operations of Algorithm 2 and Algorithm 3 can be performed while the locations are retrieved. ■

Theorem 1. *The time complexity of the SaSTL monitoring algorithm is upper-bounded by $O(|\phi|\ T_{max}\ (log(n) +$*

$|L|_{max}))$ *where* T_{max} *is the largest number of samples of the intervals considered in the temporal operators of* ϕ *and* $|L|_{max}$ *is the maximum number of locations defined by the spatial temporal operators of* ϕ.

Proof: Following Lemma 1, by considering T_{max} the worst possible number of samples that we need to consider for all possible intervals of temporal operators present in the formula, and $|L|_{max}$ for the worst possible number of locations that we need to consider for all possible intervals of spatial operators present in the formula. When there are two or more operators nested, the time complexity for one operation is bounded by $O(T_{max}\ (log(n)+|L|_{max}))$. As there are $|\phi|$ nodes in the syntax tree of ϕ, the time complexity of the SaSTL monitoring algorithm is bounded by the summation over all $|\phi|$ nodes, which is $O(|\phi|\ T_{max}\ (log(n)+|L|_{max}))$. ■

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 sub formulas. 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 sub-formula. 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 Figure 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 $\boxdot_{([0,+\infty],\mathsf{School})}\ \Box_{[a,b]}\ (\mathcal{A}^{\mathsf{op}}_{([0,d],\top])}\varphi \sim c)$ 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 $\mathsf{cost}: \Phi \times L \times G_L \to \mathbb{R}^+$, 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:

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

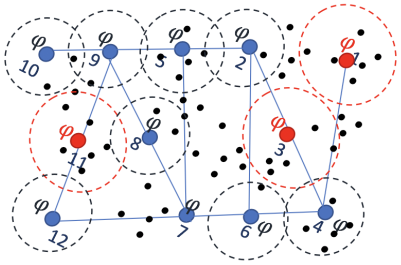


Figure 2: An example of city abstracted graph. A requirement is $\boxdot_{([0,+\infty],\mathsf{School})}\ \Box_{[a,b]}\ (\mathcal{A}^{\mathsf{op}}_{([0,d],\top)}\varphi \sim c)$ (The large nodes represent the locations of PoIs, among which the red ones represent the schools, and blue ones represent other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

Using the above function, the cost of each operation is calculated before "switch φ" (refer 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 the sub-formulas should be evaluated. The simpler sub-formula is the first to be monitored, while the more complex one is monitored only when the other sub-formula is satisfied. We update $\mathsf{monitor}(\varphi_1 \wedge \varphi_2, \omega)$ in Algorithm 4.

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

```
case φ1 ∧ φ2 do
    return Monitor(φ1, ω, t, l, G) ∧ Monitor(φ2, ω, t, l, G);
    if cost(φ1, l, G) ≤ cost(φ2, l, G) then
        if ¬ Monitor(φ1, ω, t, l, G) then
            return Monitor(φ2, ω, t, l, G);
        end
        return True;
    end
    if ¬ Monitor(φ2, ω, t, l, G) then
        return Monitor(φ1, ω, t, l, G);
    end
    return True;
end
```

With this cost function, the time complexity of the monitoring algorithm is reduced to $O(|\phi|T_{max}(log(n) + |L'|_{max}))$, where $|L'|$ is the maximal number of locations that an operation is executed with the improved parsing method. The improvement is significant for monitoring smart cities, because for most of the city requirements, $|L'|_{max} < 100 \times |L|_{max}$.

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 Figure 2, usually, it would first check the signal from location 1 with $\Box_{[a,b]}\varphi$, then location 2, and so on. At last, 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 sub-formula ($\Box_{[a,b]}\varphi$) 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 $l' \in L^l_{\mathcal{D}}$ are added to a task pool (a queue). In the mapping process, each thread retrieves monitoring tasks (i.e., for $l_i, \Box_{[a,b]}\varphi$) 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 $\mathsf{op}()$.

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

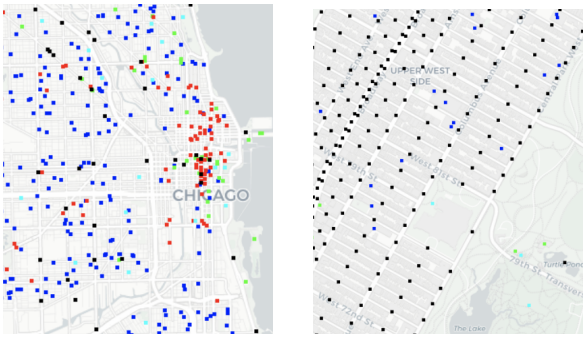
```
Function CountingNeighbours(φ, op, D, ω, t, l, G):
    begin
        paratasks = Queue();
        for l' ∈ L^l_D do
            paratasks.add(l);
        end
        results = Queue();
        for i in 1..NumThreads do
            Thread_i ← worker(φ, ω, t, G);
        end
        Wait();
        return op(results);
    end
Function worker (φ, ω, t, G):
    begin
        Real v := 0;
        if op == "min" then v := ∞; ;
        if op == "max" then v := −∞; ;
        while Num(tasks)>0 do
            l = paratasks.pop();
            moni = Monitor(φ, ω, t, l, G);
            v = op(v, moni);
        end
        results.add(v)
    end
```

Lemma 2. *The time complexity of the parallelized algorithm Monitor(ϕ, ω) is upper bounded by $O(|\phi|T_{max}(log(n) + \frac{|L|_{max}}{P}))$ 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 VI.

VI. Evaluation

We evaluate the SaSTL monitor by applying it to two city application scenarios, which are (i) runtime monitoring of city requirements in Chicago, and (ii) runtime conflict detection and resolution among smart services in a simulated New York City. Then, (iii) we compared the coverage expressiveness of SaSTL against 1000 real city requirements with STL, SSTL, and STREL. We provide the information of two application scenarios in Table III. Figure 3 presents the partial maps of two cities, where the locations of PoIs and sensors are marked.



(1) Chicago (2) New York

Figure 3: Partial Maps of Chicago and New York with PoIs and sensors annotated. (The black nodes represent the locations of sensors, red nodes represent the locations of hospitals, dark blue nodes represent schools, light blue nodes represent parks and green nodes represent theaters.)

Table III: Information of Two Application Scenarios

	Chicago	**New York**
Area (km^2)	606	60
Data Type	Real-time States	Real-time Predicted States
Data Sources	Real Sensors	Simulated Sensors and Actuators
Number of Sensor Nodes	118	10,000
Time Period	2017.01-2019.05	-
Sampling Rate	1 min	10 seconds
Domain	Environment, Public Safety	Environment, Transportation, Events, Emergencies, Public Safety
Variables	CO, NO, O_3, Visible light, Crime Rate	CO, NO, O_3, PM_x, Noise, Traffic, Pedestrian, Signal Lights, Emergency Vehicles, Accidents

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 Figure 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 the city data [1], [27] every day since January, 2017 without a state monitor. In this evaluation, we emulate the real data as it arrives in real time, i.e. assuming the city was operating with the monitor. 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 hours continuously to identify the requirement violations.

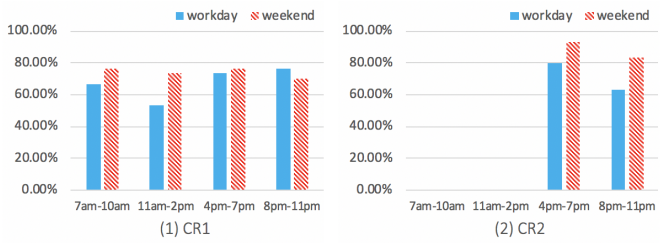


Figure 4: Requirement Satisfaction Rate during Different Time Periods in Chicago

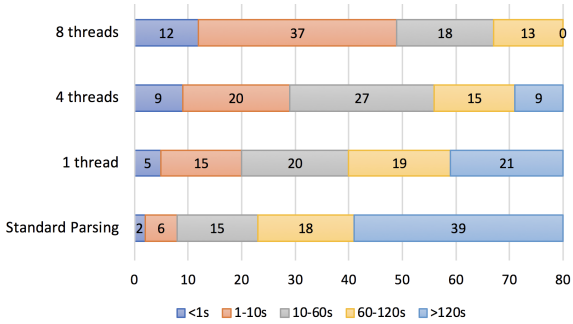


Figure 5: Number of Requirements Checked on Different Computing Time (x: the number of requirements, y: the number of threads, bars: different periods of computing time)

2) Chicago Performance: Valuable information is identified from the monitor results of different periods during a day. We randomly select 30 days of weekdays and 30 days of weekends. We divide the daytime of a day into 4 time periods and 3 hours 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 Figure 4. CR1 specifies "The average air quality within 5km of all schools should always be above *Moderate* in the next 3 hours." and is formalized as $\boxdot_{([0,+\infty),\mathsf{School})}\Box_{[0,3]}(\mathcal{A}^{\mathrm{avg}}_{([0,5],\top)}x_{\mathsf{air}} > \mathsf{Moderate})$. 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 $\boxdot_{([0,+\infty),\top)}\Box_{[0,3]}(x_{\mathsf{Crime}} = \mathsf{High} \rightarrow \mathcal{A}^{\mathrm{avg}}_{([0,3],\top)}x_{\mathsf{Light}} >= \mathsf{High})$.

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 Figure 4, 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, 8pm - 11pm) 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, 11am - 2pm has the lowest satisfaction rate of the day. The instantaneous air quality seems to be fine during 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 11am - 2pm.

3) Algorithm Performance: We count the average monitoring time taken by each requirement when monitoring for 3-hour data. Then, we divide the computing time into 5 categories, i.e., less than 1 second, 1 to 10 seconds, 10 to 60 seconds, 60 to 120 seconds, and longer than 120 seconds, and count the number of requirements under each category under the conditions of standard parsing, improved parsing with single thread, 4 threads, and 8 threads. The results are shown in Figure 5. Comparing the 1st (standard parsing) and 4th (8 threads) bar, without the improved monitoring algorithms, for about 50% of the requirements, each one takes more than 2 minutes to execute. The total time of monitoring all 80 requirements is about 2 hours, which means that the city decision maker can only take actions to resolve the violation at earliest 5 hours later. However, with the improved monitoring algorithms, for 49 out of 80 requirements, each one of them is executed within 60 seconds, and each one of the rest requirements is executed within 120 seconds. The total execution time is reduced to 30 minutes, 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 [28] 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. In this way, we use SaSTL monitor to specify requirements and check the *predicted spatial-temporal data* with the SaSTL

Table IV: 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.	$\boxdot_{([0,+\infty),\mathsf{School})} \square_{[0,30]} \ (\mathcal{A}^{\mathrm{avg}}_{([0,1],\top)} x_{\mathsf{Noise}} < 50)$
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.	$\boxdot_{([0,+\infty],\top)}(\mathsf{Accident} \rightarrow \mathcal{C}_{([0,5],\mathsf{Hospital})}(\square_{[0,60]}(\mathcal{A}^{\mathrm{avg}}_{([0,2],\top)} x < \mathsf{Congestion})) > 0)$
NYR3	If there is an event, the max number of pedestrians waiting in 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.	$\mathcal{C}^{\mathrm{avg}}_{([0,+\infty),\top)}(\square_{[0,60]}(\mathcal{A}^{\max}_{([0,1],\top)} x_{\mathsf{PMx}} < \mathsf{Moderate})) > 0.9$
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.	$\boxdot_{([0,+\infty),\top)}(\mathsf{Accident} \rightarrow (\mathcal{A}^{\mathrm{avg}}_{([0,500],\top)} x_{\mathsf{traffic}} < \mathsf{Moderate} \wedge \mathcal{A}^{\max}_{([0,500],\top)} x_{\mathsf{traffic}} < \mathsf{Safe}) \mathcal{U}_{[0,60]} \neg \mathsf{Accident})$

formulas. If there exists a requirement violation within the future horizon, a conflict is detected. To resolve the conflict, it provides several possible resolution options and predicts the outcome of all these options, which are verified by the SaSTL monitor. It finds the best option without a violation, otherwise, it provides the trade-offs between a few potential resolution options to the city manager through monitoring the predicted traces. Details of the resolution are not the main part of this paper and we thank the original authors for making the solution available to the authors of the present paper.

We set up a smart city simulation of New York City using the Simulation of Urban MObility (SUMO) [29] with the real city data [30], on top of which, we implement 10 smart 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 states from the domains of environment, transportation, events and emergencies are generated from about 10000 simulated nodes. Please see Table III for the variables.

We use the STL 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 in three control sets, one without any monitor, one with the STL monitor and one with the SaSTL monitor. For the first set (no monitor), there is no requirement monitor implemented. For the second one (STL monitor), we specify NYR1 to NYR5 using STL without aggregation over multiple locations, i.e., NYR1 is a set of requirements on many locations and each location has a temporal requirement. For example, NYR1 is specified as $\varphi_{l_i} = \square_{[0,t]}(x_{\mathsf{air}} > \mathsf{Moderate})$, where $l_i \in L$, L is a set of sensors within the range. The setting is the same for the rest of the requirements. For the third one (SaSTL monitor), five examples of different types of real-time requirements and their formalized SaSTL formulas are given in Table IV.

2) NY City Performance: The results are shown in Table V. 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, 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. To be noted, the number of violations detected is the total number of safety requirements violated, rather than the number of conflicts. Many times there is more than one requirement violated in one conflict.

Table V: Comparison of the City Performance with the STL Monitor and the SaSTL Monitor

	No Monitor	STL Monitor	**SaSTL Monitor**
Number of Violation	Unknown	219	**173**
Air Quality Index	67.91	51.22	**40.18**
Noise (db)	73.32	49.27	**41.42**
Emergency Waiting Time (s)	20.32	12.93	**11.88**
Vehicle Waiting Number	22.7	18.2	**12.6**
Pedestrian Waiting Time (s)	190.2	130.9	**61.1**
Vehicle Waiting Time (s)	112.12	70.98	**59.22**

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 V, the number of violations detected by SaSTL Monitor is 46 less than the STL monitor. On average, the framework of conflict detection and resolution with the SaSTL monitor improves the air quality by 21.6% and 40.8% comparing to the one with the STL monitor and without a monitor, respectively. It improves the pedestrian waiting time by 16.6% and 47.2% comparing to the STL monitor and without a monitor, respectively.

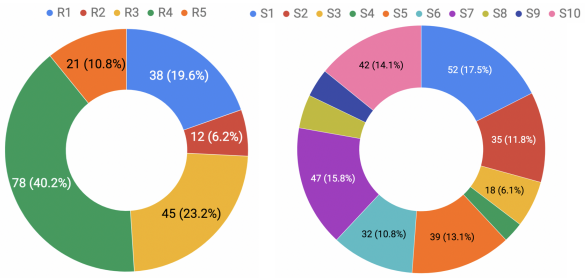
Second, *the SaSTL monitor reveals the real city issues, helps refine the safety requirements in real time and supports improving the design of smart services.* We also compare the number of violations on each requirement. The results (Figure 6 (1)) help the city managers to measure city's performance 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 necessary. For example, in our 30 days simulation, apparently, NYR4 on air pollution is the one requirement that is violated by most of the smart services. Similarly, Figure 6 (2) 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 4 threads and (4) using 8 threads. The results are shown in Table VI.

Table VI: 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**



(1) Requirements (2) Smart Services

Figure 6: Distributions of the violations over requirements and smart services

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 seconds to 140.29 seconds (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 reduced to 26.12 seconds with 8 threads while 140.29 seconds with single thread (about 5 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 5 requirements is reduced from 2678.5 seconds to 100.85 seconds. In the real world, when multiple requirements are monitored simultaneously, the improvement is extremely important for real-time monitoring.

C. Coverage Analysis

We compare the specification coverage on 1000 quantitatively-specified real city requirements between STL, SSTL, STREL and SaSTL, the results are shown in Figure 7. 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.

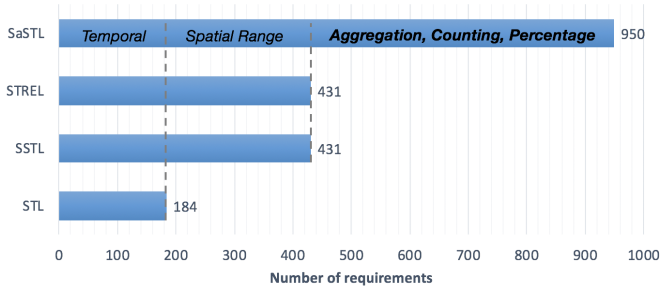


Figure 7: Comparison of the Specification Coverage on 1000 Real City Requirements

As shown in Figure 7, STL is only able to specify 184 out of 1000 requirements out, 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. Secondly, 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 dynamic graph and check requirements reachability, which is very useful in applications like wireless sensor networks, but not common in smart city requirements; Fourth, for the rest of the requirements (467 out of 1000) that can only be specified by SaSTL contains a various set of locations and the aggregation over spatial ranges.

VII. Related Work

Monitoring spatial-temporal properties over CPS executions was first proposed in [31] where the author has introduced the notion of spatial-temporal event-based model for

CPS. In such model events triggered by actions, exchange of messages or a physical changes, are labeled with time and space stamps and further processed by a monitor. In [32] this concept is further elaborated, developing a spatial-temporal event-based model where the space is represented as a 2D Cartesian coordinate system with location points and location fields. The approaches described in [31], [32] provide an algorithmic framework enabling a user to develop manually a monitor. However, they do not provide any spatio-temporal logic language enabling the specification and the automatic monitoring generation.

In the field of *collective adaptive systems* [33], other mathematical structures such as *topological spaces*, *closure spaces*, *quasi-discrete closure spaces* and *finite graphs* [13] have been considered to reason about spatial relations, such as *closeness* and *neighborhood*. In our previous works [34], [35], we have also studied *quad-trees* to reason about fractal-like spatial relations or spatial superposition properties in a grid, such as electrical spiral formation in cardiac tissues [36] or power management requirements in a smart grid [34]. Despite these models are suitable for offline and centralised monitoring of model-based simulations, they do not scale well for the runtime monitoring of spatially distributed CPS.

Several logic-based formalisms have been proposed to specify the behavior and the spatial structure of concurrent systems [37] and for reasoning about the topological [38] or directional [39] aspects of the interacting entities. In topological reasoning [38], the spatial objects are sets of points and the relation between them is preserved under translation, scaling and rotation. In directional reasoning, the relation between objects depends on their relative position. These logics are usually highly computationally complex [39] or even undecidable [40]. Monitoring spatial-temporal behaviors has started to receive more attention only recently with SpaTeL [34] and SSTL [13].

The *Spatial-Temporal Logic* (SpaTeL) is the unification of *Signal Temporal Logic* [12] (STL) and *Tree Spatial Superposition Logic* (TSSL) introduced in [35] to classify and detect spatial patterns. TSSL reasons over quad trees, spatial data structures that are constructed by recursively partitioning the space into uniform quadrants. The notion of superposition in TSSL 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 [36] that generally characterize the patterns emerging in nature. 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.

Another important work to mention is VOLTRON [41], an open-source *team-level programming* system for drone's collaborative sensing. VOLTRON provides special programming constructs to reason about time and space and allows users to express sophisticated collaborative tasks without exposing them to the complexity of concurrent programming, parallel execution, scaling, and failure recovery. The spatial constructs are limited to operate on a set of locations of a given geometry (that the user need to specify). The system is suitable more for programming than for monitoring. For example, it does not allow to quantify how much the current CPS execution is close to violate a given requirement.

VIII. CONCLUSION

In this paper, we present a novel Spatial Aggregation Signal Temporal Logic (SaSTL) to specify and to monitor real-time safety requirements of smart cities at runtime. We develop an efficient monitoring framework that optimizes the requirement parsing process and can check in parallel a 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. real-time, interval), spatial (e.g., PoIs, range) and their complicated relations (e.g. always, everywhere, 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. We believe it is a valuable step towards developing a practical smart city monitoring system even though there are still open issues for future work. Furthermore, SaSTL monitor is more than a smart city monitor, it can also be easily generalized and applied to monitor other large-scale IoT deployments with a large number of sensors and actuators across the spatial domain at runtime efficiently.

ACKNOWLEDGMENT

Supported in part by NSF NRT Grant 1829004.

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 *Proceedings of the 2nd International Workshop on Science of Smart City Operations and Platforms Engineering*. ACM, 2017, pp. 26–33.

[2] New York Times, "IBM takes 'smarter cities' to rio de janeiro," 2012.

[3] Cisco, "Smart+connected operations center," 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 Transactions on Cyber-Physical Systems*, vol. 4, no. 2, pp. 1–28, 2019.

[5] Y. Yuan, D. Zhang, F. Miao, J. A. Stankovic, T. He, G. Pappas, and S. Lin, "Dynamic integration of heterogeneous transportation modes under disruptive events," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2018, pp. 65–76.

[6] 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.

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

[8] S. Sheng, E. Pakdamanian, K. Han, B. Kim, P. Tiwari, I. Kim, and L. Feng, "A case study of trust on autonomous driving," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 4368–4373.

[9] 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.

[10] 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 *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*. ACM, 2015, pp. 189–198.

[11] M. Ma, J. A. Stankovic, and L. Feng, "Cityresolver: a decision support system for conflict resolution in smart cities," in *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*. IEEE Press, 2018, pp. 55–64.

[12] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Proc. FORMATS*, 2004.

[13] L. Nenzi, L. Bortolussi, V. Ciancia, M. Loreti, and M. Massink, "Qualitative and quantitative monitoring of spatio-temporal properties," in *Runtime Verification - 6th International Conference, RV 2015*, vol. 9333. Springer, 2015, pp. 21–37.

[14] E. Bartocci, L. Bortolussi, M. Loreti, and L. Nenzi, "Monitoring mobile and spatially distributed cyber-physical systems," in *MEMOCODE 2017: the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design*. ACM, 2017, pp. 146–155.

[15] NYC.gov, "Emissions from transportation, nyc environment protection," 2019. [Online]. Available: https://www1.nyc.gov/html/dep/html/air/emissions_from_transportation.shtml

[16] District of Columbia Municipal Regulations and D. of Columbia Register, "Air quality - motor vehicular pollutants, lead, odors, and nuisance pollutants," 2016.

[17] 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," in *Restricting the use of bus lanes by sight-seeing buses*. The New York City Council, 2019.

[18] NYC Environment Protection, "Use of heating oil remaining in tanks." The city of New York, 2019.

[19] United States Environmental Protection Agency, "Residential energy efficiency," in *Energy Resources for State and Local Governments*. The city of New York, 2019.

[20] LA Sec 111.03. Minimum Ambient Noise Level, "Official city of los angeles municipal code," 2016.

[21] Hong Kong, "Guide to indoor air quality management in hong kong regional offices and public places," in *Guide to Indoor Air Quality Management*, 2019.

[22] NYC.gov, "Stopping, standing or parking prohibited in specified places," in *New York Public Law*, 2016.

[23] Beijing Emergency Agency, "Pre-hospital medical emergency regulations," 2016.

[24] Beijing Government, "Safety management for kindergarten, primary and secondary school," 2016.

[25] A. Donzé, T. Ferrere, and O. Maler, "Efficient robust monitoring for STL," in *International Conference on Computer Aided Verification*. Springer, 2013, pp. 264–279.

[26] G. S. Lueker, "A data structure for orthogonal range queries," in *19th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 1978, pp. 28–34.

[27] City of Chicago, "Crimes of Chicago - one year prior to present," 2018.

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

[29] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "Sumo–simulation of urban mobility: an overview," in *Proceedings of SIMUL 2011*. ThinkMind, 2011.

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

[31] C. L. Talcott, "Cyber-physical systems and events," in *Software-Intensive Systems and New Computing Paradigms - Challenges and Visions*, ser. LNCS. Springer, 2008, vol. 5380, pp. 101–115.

[32] Y. Tan, M. C. Vuran, and S. Goddard, "Spatio-temporal event model for cyber-physical systems," in *2009 29th IEEE International Conference on Distributed Computing Systems Workshops*. IEEE, 2009, pp. 44–50.

[33] V. Ciancia, D. Latella, M. Loreti, and M. Massink, "Spatial logic and spatial model checking for closure spaces," in *Proc. of SFM 2016*, ser. LNCS, vol. 9700. Springer, 2016, pp. 156–201.

[34] I. Haghighi, A. Jones, J. Z. Kong, E. Bartocci, G. R., and C. Belta, "SpaTeL: A Novel Spatial-Temporal Logic and Its Applications to Networked Systems," in *Proc. of HSCC*, 2015.

[35] E. A. Gol, E. Bartocci, and C. Belta, "A formal methods approach to pattern synthesis in reaction diffusion systems," in *Proc. of CDC*, 2014.

[36] 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

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

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

[39] D. Bresolin, P. Sala, D. D. Monica, A. Montanari, and G. Sciavicco, "A decidable spatial generalization of metric interval temporal logic," in *2010 17th International Symposium on Temporal Representation and Reasoning*, 2010, pp. 95–102.

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

[41] L. Mottola, M. Moretta, K. Whitehouse, and C. Ghezzi, "Team-level programming of drone sensor networks," in *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems, SenSys '14, Memphis, Tennessee, USA, November 3-6, 2014*. ACM, 2014, pp. 177–190.