Spatio-temporal Model-checking of Vehicular Movement in Public Transport Systems

Citation for published version:
Ciancia, V, Gilmore, S, Grilletti, G, Latella, D, Loreti, M & Massink, M 2018, 'Spatio-temporal Model-checking of Vehicular Movement in Public Transport Systems', *International Journal on Software Tools for Technology Transfer*. https://doi.org/10.1007/s10009-018-0483-8

Digital Object Identifier (DOI):
10.1007/s10009-018-0483-8

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
International Journal on Software Tools for Technology Transfer

Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

Spatio-temporal Model-checking of Vehicular Movement in Public Transport Systems

Vincenzo Ciancia · Stephen Gilmore · Gianluca Grilletti · Diego Latella · Michele Loreti · Mieke Massink

Received: date / Accepted: date

Abstract We present the use of a novel spatio-temporal model-checker to detect problems in the data and operation of a collective adaptive system. Data correctness is important to ensure operational correctness in systems which adapt in response to data. We illustrate the theory with several concrete examples, addressing both the detection of errors in vehicle location data for buses in the city of Edinburgh and the undesirable phenomenon of "clumping" which occurs when there is not enough separation between subsequent buses serving the same route. Vehicle location data is visualised symbolically on a street map, and categories of problems identified by the spatial part of the model-checker are rendered by highlighting the symbols for vehicles or other objects that satisfy a property of interest. Behavioural correctness makes use of both the spatial and temporal aspects of the model-checker to determine from a series of observations of vehicle locations whether the system is failing to meet the expected quality of service demanded by system regulators.

Keywords Spatio-temporal model checking · Collective-adaptive systems · Smart transportation

This work is partially supported by the EU project *QUANTICOL: A Quantitative Approach to Management and Design of Collective and Adaptive Behaviours*, 600708.

V. Ciancia, D. Latella, M. Massink
Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", Consiglio Nazionale delle Ricerche, Italy. E-mail: {vincenzo.ciancia,diego.latella,mieke.massink}@isti.cnr.it.

M. Loreti
Dipartimento di Statistica, Informatica, Applicazioni "G. Parenti", Università di Firenze, Italy. E-mail: michele.loreti@unifi.it.

G. Grilletti
Institute for Logic, Language and Computation, University of Amsterdam, The Netherlands. E-mail: grilletti.gianluca@gmail.com.

Stephen Gilmore
Laboratory for Foundations of Computer Science, University of Edinburgh, Edinburgh, Scotland. E-mail: stephen.gilmore@ed.ac.uk.

Mathematics Subject Classification (2010) MSC 68N30 - Mathematical aspects of software engineering · 68Q60 - Specification and verification · 03B70 - Logic in Computer Science

1 Introduction

Users, operators and regulators of managed services want to have systems which behave correctly across a wide range of conditions. Operational correctness is monitored by regulators and it is the responsibility of system operators to try to ensure that system operation lies within allowable bounds as often as possible in practice. A typical example of operational correctness is to guarantee sufficient separation in time between subsequent buses in so-called "frequent" bus services that do not follow a fixed time table.

Adaptive systems depend crucially on real-time data collection subsystems which allow them to reflect on their operation, detect problems in their service, and report these problems. These data collection systems are often built from physical components such as sensors and receivers which have limits to their engineering, meaning that they sometimes deliver inaccurate measurement data.

In a small-scale supervised system where the measurement data is interpreted by a human before any action is taken, erroneous data might not be very problematic because the data can be checked before any action is taken as a consequence. In a collective adaptive system however, the demands of scale and responsive adaptivity may mean that there is no human oversight of the data before action is taken as a consequence of the data received.

Physical components can only deliver accurate information up to their physical limits, and size and weight considerations often severely curtail the amount of processing which can be done on an embedded device. These practicalities mean that the responsibility for dealing with erroneous data lies with the system itself, and that it must make efforts to automate the process of data checking and cleaning before acting in response to data received. The task of achieving operational correctness comes after the task of achieving data correctness. In this paper we address examples of both, exploiting novel purely spatial and combined spatio-temporal model-checking techniques.

Knowing where the system assets are located is often of critical importance for correct functionality. In a modern public transport system many aspects of well-regulated operation depend on accurate fleet management, supported by an automatic vehicle location (AVL) system. AVL data drives other systems such as bus arrival prediction systems and allows operators to produce quantitative measures of service quality.

In this paper, we use spatio-temporal model checking to discover problems related to spatial and operational aspects of a collective adaptive system[1]. We chose bus networks as a prototypical example of a collective adaptive system in which analysis is strictly tied to geographical features (see for instance [20] for an example of how collective adaptive behaviour may be exploited in bus networks). Specifically, we assess correctness of geographic data that may come from agents

[1] A collective adaptive system consists of multiple cooperating components with decentralised control – these may be human or non-human — and adapts itself to unexpected problems arising from the environment in which it operates.

in the system. GPS data received from buses is related to a map, in the form of a digital image, that is meant to incorporate all the information that may be relevant to data correctness. The relevance of specific features (e.g., size of streets and crossings, morphological considerations such as bend angles, position of buildings or squares, pedestrian areas, rivers, etc.) may only be known at analysis time, therefore making it difficult to use abstractions of a map, such as the graph of streets and crossings, that may "abstract too much" and omit useful information. In our approach, instead, analysis is designed through declarative logical means, that make explicit use, on a case-by-case basis, of the visual features of the map that are needed.

We consider both purely spatial aspects, namely, data correctness, and spatio-temporal issues, related to the behaviour of buses in the network. The spatial verification problem we consider is that of determining whether or not the AVL data received about vehicles in the fleet indicates an error condition. We show how to identify such problems in the data by the means of a recently developed *spatial* model-checker [14]. Using a formal language (namely, the *Spatial Logic of Closure Spaces* SLCS, featuring Boolean and *spatial* operators), requirements on spatial data can be mathematically formalised, and the model-checker can verify data correctness in a fully automated way. The use of logic formulas makes the approach declarative, in that subtle aspects of the analysis can be changed by editing the (very short) formulas, without needing to modify analysis algorithms. Depending on the intended result, the methods presented here could either be used for on-line or off-line data processing. In on-line data processing the smart infrastructure within the system would attempt to classify problems in real-time, alerting operators to problems as they occur. In off-line data processing the infrastructure would attempt to classify problems with the benefit of hindsight, providing plausible retrospective explanations for events so that the service operators can review their service and provide reports for service regulators, and use the insights gained to improve the service at the next offering (for example, the following day).

Taking different sections from the data allows one to address different spatial model-checking problems. We could choose to project an instantaneous snapshot of the current location of all of the buses in the fleet (a "satellite view", allowing one to see all buses at one time). Alternatively we could choose to project the journey of a particular bus in the fleet as a function of time (a "passenger view", allowing one to see one bus at all times).

The satellite view allows one to raise issues about congestion and adjacency; the passenger view allows one to ask questions about journeys and routes. In this paper we will use single satellite views when we are investigating data correctness ("Is this bus really so far off-route?") and sequences of successive satellite views when we are investigating operational correctness ("Is this bus catching up with the one in front?").

Besides the purely spatial aspects, we also illustrate the use of model-checking to detect operational problems of the bus network. We use the spatio-temporal model-checker developed in [13], extending the spatial model checker of [14,15] with temporal features. The temporal fragment is developed as a variant of the well-known *Computation Tree Logic* (CTL). To exemplify the descriptive capabilities of such spatio-temporal logics, we show how spatio-temporal model checking can be used to assess and validate possibly non-deterministic policies for mitigating operational problems in adaptive systems. In particular, we use as an example the problem of *clumping* (also called *bunching*, *convoying*, or *platooning*) on a bus route.

On bus routes with frequent service, the most relevant metric is not adherence to a pre-defined timetable, but rather maintaining a reasonable separation (known as *headway*) between different buses passing by each stop. This is needed to maintain a good distribution of passengers across buses and to provide passengers with a predictable service. Clumping is the phenomenon where two or more buses serving the same route get very close to each other. The problem is caused by a negative feedback loop (as first explained by Newell and Potts in [34]) where buses running late find ever more people waiting at the bus stop, causing further delay, and buses following such late buses encounter few passengers and no delays, causing them to catch up with the bus in front. Clumping may therefore occur independently of traffic conditions. Clumping is the most frequent cause of user dissatisfaction in frequent bus services [20]. Formal specification of the headway requirement may be interpreted in different ways. We discuss how such different interpretations correspond to different spatio-temporal logic formulas, that can be machine-checked against models of the bus network.

Finally, we provide some preliminary ideas on how spatio-temporal model-checking may be used to analyse and validate the effect of policies aimed at correcting operational problems, illustrating this on an example of bus clumping. Bus clumping, and more general issues related to headway and slack time of buses, are the subject of an active research field; see for instance [29, 28, 20, 19, 37, 44, 38]. Promising results are obtained by bus holding strategies such as those described in [29, 28, 20]. Such strategies can be implemented by letting buses wait for some additional time, with a duration that depends on the headway, at pre-defined control points in order to maintain a constant headway between subsequent buses. These strategies have been shown to be able to self-regulate the headway of buses also in the presence of perturbations in the system (e.g. buses breaking down and needing to stop for repairs, changing traffic conditions, or temporary changes in bus routes) [29, 28, 20]. In our example, we derive a branching time spatio-temporal model from a single series of snapshots of bus positions projected onto a city map, that takes into account the possibility of buses waiting a short fixed period of time along their path for one or more times. By augmenting an existing GPS trace, featuring clumping, with choice points where a given bus may be requested to wait for a short amount of time, the model checker is then used to analyse the effects of clumping mitigation strategies. The idea is that a more elaborate approach along these lines could provide us with an alternative framework for validation of some bus holding strategies in the context of actual traces of bus movement obtained in real cities. Our method could be exploited further, e.g. in order to compute the minimal waiting time needed to guarantee that no incorrect execution traces exist, or in order to compute probabilities by exploiting statistical model-checking techniques. This is left as possible future work.

This paper is organised as follows. In Section 2 we address pure spatial model-checking, without the temporal dimension, providing an informal introduction and examples. Section 3 presents the various categories of data issues and how they are visualised on a street map. Section 4 illustrates how these data issues can be identified exploiting spatial model-checking on a portion of a city map. In Section 5 we extend spatial model-checking with a temporal dimension introducing a spatio-temporal logic. Section 6 addresses some bus operational issues concerning the problem of clumping, both in time and in space. In Section 7 we show how to use spatio-temporal model-checking to assess the effect of correction strategies to alleviate clumping. We conclude the paper by an overview of related work in Section 8 and conclusions and outlook on future work in Section 9.

2 Spatial model-checking

In its original conception, model-checking [3] is a technique that was developed for the formal verification of properties of distributed and concurrent systems. It takes a formal model of the system and a property of interest, usually expressed as a temporal logic formula (see for instance [6,23,17]), and then checks, in an automatic way, whether the model satisfies the property. Properties of the temporal evolution of a system are considered, but properties of (physical) space typically are not. In recent work [14,15] by some of the co-authors of this paper, a *spatial* model-checking approach has been developed. This technique builds on *spatial logics*, that is, topological interpretations of modal logics [41], dating back to Tarski.

Of particular interest to us is a spatial *surrounded* operator – inspired by the temporal *weak until* operator – that first appeared for topological spaces in [1]. In [14], the operator has been studied together with a model-checking algorithm in the more general setting of *closure spaces.* The spatial variant of the connective can be used to define conditional reachability properties in a spatial setting. The logical operators have been lifted to closure spaces, so that they can also be used on discrete, graph-based structures, which include digital images, and various kinds of geographic maps. In this paper, we consider specific graphs, namely digital images, seen as regular grids, where the nodes are the points in the image and where edges connect each node to the adjacent nodes in the north, south, east and west direction.

In [14], an efficient proof-of-concept model-checker for the Spatial Logic of Closure Spaces (SLCS) has been implemented. The tool is able to interpret spatial logic formulas on general finite graphs (more precisely on quasi discrete closure spaces). In the special case in which such graphs are regular grids representing digital images a graphical understanding of the meaning of formulas can be provided, and thus an immediate form of counterexample visualisation. Points that satisfy a particular formula can be visualised by a colour of choice on an image. In this paper, we use the spatial model-checker for some very pragmatic purposes that are concerned with the automatic identification of potential errors in AVL data. The approach we follow is to project the AVL data, including the exact position of the vehicles, on an existing digital map of the relevant geographical area. We then use the spatial model-checker to identify and highlight regions of interest on this map.

The spatial logic consists of a very small number of basic operators that, in turn, are used to define a number of useful derived operators. Among the basic operators are the standard Boolean constants (*true* $\top$, *false* $\bot$) and connectives (*negation* $\neg$, *disjunction* $\vee$ and *conjunction* $\wedge$), the *near* operator $\mathcal{N}$ and the *surrounded* operator $\mathcal{S}$. The near operator has its origin in the *closure* operator of topological spatial logics. More precisely, models of the logic are closure spaces. A closure space is a pair $(X, \mathcal{C})$, consisting of a set of points X and a function $\mathcal{C}$, from 2^X to 2^X, such that, for all sets $A, B \subseteq X$ we have: $\mathcal{C}(\emptyset) = \emptyset$, $A \subseteq \mathcal{C}(A)$ and $\mathcal{C}(A \cup$

$B) = \mathcal{C}(A) \cup \mathcal{C}(B)$. Note that the closure operator is not required to be idempotent; the class of closure spaces whose closure operator *is* idempotent coincides with topological spaces (via the so-called *Kuratowski definition* of topological spaces). Topological spaces are therefore a proper subclass of closure spaces.

In our logic, the near operator is denoted by $\mathcal{N}$. The formula $\mathcal{N}\,\phi$ is satisfied by a point x in space if x satisfies ϕ or it is a direct neighbour of a point satisfying ϕ. The surrounded operator $\mathcal{S}$ is a spatial version of the temporal *weak until* operator (also called "unless" in the literature). A point x in space satisfies the formula $\phi\,\mathcal{S}\,\psi$ whenever it is included in an area A consisting of points satisfying formula ϕ and there is "no way out" from A unless passing through points in an area B that satisfies ψ, see Figure 1. Finally, we assume a set of basic propositions, which in our specific case identify the colour of a pixel in the digital map[2].

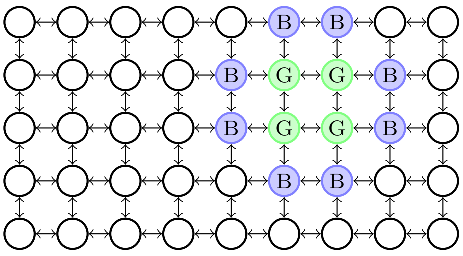


Fig. 1: The green nodes satisfy green surrounded by blue $((G)\,\mathcal{S}\,(B))$

We will use a few derived operators (see Figure 2) in the properties shown in the next section. The first derived operator is the spatial reachability operator $\phi\,\mathcal{R}\psi$. It is defined in terms of the surrounded operator as $\phi\,\mathcal{R}\psi \triangleq \neg((\neg\psi)\,\mathcal{S}\,(\neg\phi))$ and, abstracting from some details, it identifies those points from which ψ can be reached passing only by points satisfying ϕ. Further examples of derived operators can be found in [15]. In this paper, we use: the "touch" operator $\phi\mathcal{T}\psi \triangleq \phi \wedge ((\phi \vee \psi)\,\mathcal{R}\psi)$ denoting all the points laying in an area, say A, such that each point in A directly touches either one point satisfying ψ or another point in A, and all points in A further satisfying ϕ; the "everywhere" operator $\mathcal{E}\phi \triangleq \phi\,\mathcal{S}\,\bot$, that, when the considered space is connected (this is the case of images), is a "global" assertion, either denoting all points of the space, when all points satisfy ϕ, or no point otherwise; the "somewhere" operator $\mathcal{F}\phi \triangleq \neg\mathcal{E}\neg\phi$, which denotes all points of the space, when there is at least one point satisfying ϕ, and no point otherwise.

We briefly recall the formal semantics of SLCS that was first presented in [14] using slightly different notation.

Definition 1 Given a (finite or countable) set of *atomic propositions* P, and a valuation function $\mathcal{V} : P \to 2^X$, satisfaction is defined in a SLCS *model* $\mathcal{M} =$

[2] Note that in general any kind of basic propositions may be chosen. For example they may be values representing concentrations of chemical substances or probabilities as shown in the various case studies presented in [13,16].

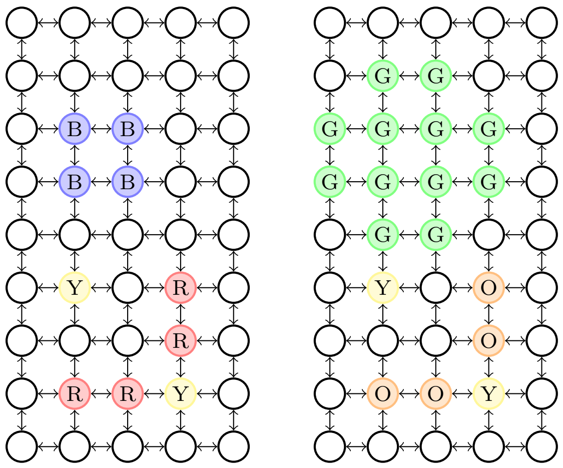


Fig. 2: Example of the interpretation of spatial formulas (left: original model; right: superimposed results of interpretation). The green (G) nodes (on the right) satisfy the closure of the blue (B) ones (shown in the picture on the left), and the orange (O) nodes (on the right) satisfy the formula $(R)\,\mathcal{T}\,(Y)$ applied to the left figure, reaching yellow (Y) nodes only from and via red (R) ones. All the nodes in both figures satisfy $\mathcal{E}\,(\neg(K))$, as there is no node satisfying K, whereas all the nodes on the left, and no node on the right, satisfy $\mathcal{F}\,(B)$.

$((X, \mathcal{C}), \mathcal{V})$ at point $x \in X$ as follows:

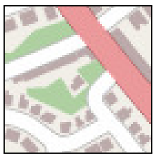
$$\begin{array}{lll}
\mathcal{M}, x \models \top & & \\
\mathcal{M}, x \models p \in P & \iff & x \in \mathcal{V}(p) \\
\mathcal{M}, x \models \neg\Phi & \iff & \mathcal{M}, x \not\models \Phi \\
\mathcal{M}, x \models \Phi \vee \Psi & \iff & \mathcal{M}, x \models \Phi \text{ or } \mathcal{M}, x \models \Psi \\
\mathcal{M}, x \models \mathcal{N}\,\Phi & \iff & x \in \mathcal{C}(\{y \in X | \mathcal{M}, y \models \Phi\}) \\
\mathcal{M}, x \models \Phi\ \mathcal{S}\ \Psi & \iff & \exists A \subseteq X . x \in A \wedge \forall y \in A . \mathcal{M}, y \models \Phi \wedge \\
& & \quad \wedge \forall z \in \mathcal{B}^+(A) . \mathcal{M}, z \models \Psi
\end{array}$$

In the above definition $\mathcal{B}^+$ denotes the *closure boundary* which is defined as $\mathcal{B}^+(A) = \mathcal{C}(A) \setminus A$.

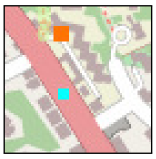
Roughly speaking, the model-checker takes as input a finite model, and a formula ϕ, and returns all the points of the given closure space that satisfy ϕ. The function that computes the satisfaction set is defined inductively on the structure of ϕ following a bottom-up approach. The original part of the algorithm concerns the surrounded operator $\phi\,\mathcal{S}\,\psi$. The algorithm performs a backwards search from the set of nodes that do not satisfy ϕ or ψ but that can be reached in one step from such nodes. The algorithm terminates in $\mathcal{O}(k \cdot (|X| + |R|))$, where k is the number of operators in the formula, $|X|$ the number of nodes and $|R|$ the number of edges in the graph [14, 15].

3 Categories of data issues in the case study

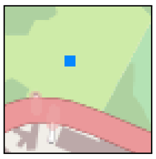
In this section we introduce the features of the digital image representing a city map and see the effect of the spatial properties which we evaluate on such maps. Figure 3 explains our conventions for representing buses, bus stops and the progress in time through observations on the map.



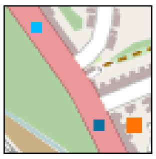
We use an OpenStreetMap representation for the map from which the names of streets have been removed for clarity. Main streets are pink, side streets are white.



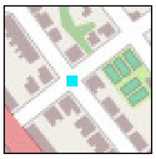
Buses are represented by blue squares on the map in a range of shades of blue. Bus stops are represented by orange squares.



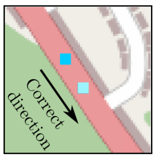
Errors in GPS data show up as bus observations which are not on any road.



In this example, the colour of a bus darkens slightly from one observation to the next. In this case the observation nearer to the bus stop has a later timestamp than the observation which is further away from the bus stop.



Diverted buses are seen as being on side streets which are off the main bus route.



Unexpected changes in position are significant. If a later timestamped observation shows the bus less further along its route then this usually indicates a data error because buses rarely reverse along a road.

Fig. 3: Representations of buses, roads and bus stops on maps

We have several categories of data issues to distinguish:

- Plausible: The bus is positioned on a road and it is a road where we would expect to see a bus. Nothing about this data point seems problematic; adaptive behaviour based on this data observation would seem to be acting on good data.
- Implausible: This data point seems suspicious; the bus is positioned in an area of the city where we would not normally expect to see a bus (such as in a field,

or a wood, or a pond). Unsupervised adaptive behaviour based on this data would be inadvisable.

- Possible: This data point has a bus positioned on a road but it is a side street when we were expecting to see the bus on a main road. The data is not implausible but it indicates that an unexpected event has perhaps occurred (a road closure, a traffic accident, or a diversion caused for another reason). Adaptive behaviour based on this data observation might be appropriate here.
- Problematic: This data point shows a bus on the expected route but it is less far along the route than previously reported. It is likely that either this data point is putting the bus behind its current position, or the previous data point put it ahead of its current position (or, possibly, the bus is reversing). Adaptive behaviour based on this data should be delayed until the uncertainty about which point is incorrect is resolved.

4 Identifying data issues using model-checking

The model-checker has been used to analyse a coloured digital image, representing a portion of the map of the city of Edinburgh, augmented with squares filled in special colours, denoting particular kinds of entities, as described in Section 3. Logic formulas are used in this example to detect problems in the data. We also show how to detect other features of interest of bus positions, and of roads. We shall now describe the formulas we use more in detail; these are interpreted on the model depicted in Figure 4. The red circles, as well as the yellow balloon with text, have been superimposed on the images and are not part of input or output of the tool (this is done for readability, also in Sections 6-7).

Atomic predicates, when working on images, are actually colour ranges. Because of this, we can analyse an image directly, without the need for additional meta-data. For this example, we selected 14 data points to represent buses[3], represented by different blue squares. The shade of blue depends upon the time at which the bus was in the given position. The shade ranges from light to dark, where lighter shades precede darker ones in time. Then, using atomic predicates on colours and colour ranges, we defined various basic formulas, among which: formula `bus` representing all the bus positions (using a colour range); formulas `pos1`, ..., `pos14`, representing the separate bus positions; formulas identifying streets (`street`) and main streets (`mainStreet`).

The spatial model-checker accepts an image as input and produces a transformed image, where the points that satisfy the formula of interest are coloured in a user-defined colour. In the current example, the model presented to the model-checker is the map image with reported bus positions marked (using blue squares) and the positions of bus stops marked (using orange squares). For example, positions of diverted buses can be repainted in red and positions of off-road buses can be repainted in violet.

In the following examples, we use the concrete input syntax of the SLCS model-checker `topochecker` (described in more detail in Section 5), where the logical symbols $\top$, $\bot$, $\neg$, $\wedge$, $\vee$, $\mathcal{N}$, $\mathcal{S}$ are denoted by `TT`, `FF`, `!`, `&`, `|`, `N`, and `S`, respectively.

[3] In this paper, the points have been selected artificially, in order to present a clear working example, but use of the spatial model-checker does not differ when dealing directly with the vehicle location data supplied by Lothian Buses.

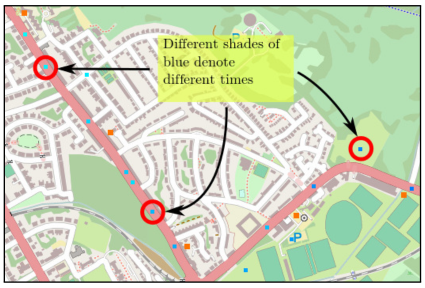


Fig. 4: Input model. Blue squares are bus positions; their order in time is described by the increasing darkness. Orange squares are the bus stops.

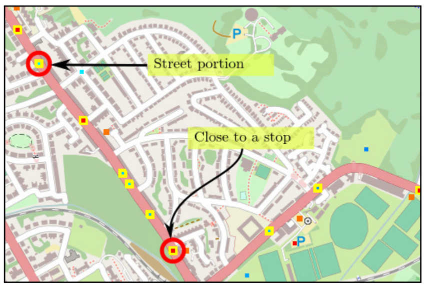


Fig. 5: Positions close to a stop are repainted in red, the areas of the road surrounding all bus positions are repainted in yellow.

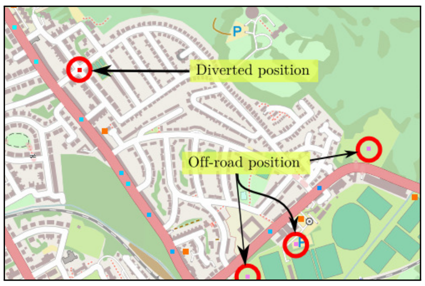


Fig. 6: Diverted positions (neither off road, nor on a main street) are repainted in red, off-road positions are repainted in violet.

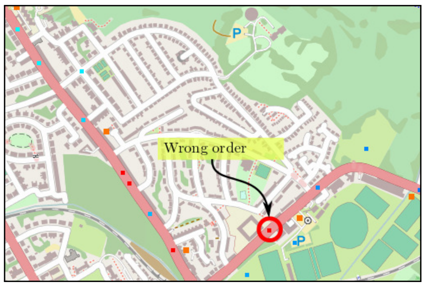


Fig. 7: Positions that are found to be out of order (i.e., not between the previous and next position) are repainted in red.

Spatial features of data points In Figure 5 we show the result of identifying a portion of the main street for each position (depicted in yellow); this is done using the formula:

```
Let streetPortion(b) =
    mainStreet & (N^3 b)
    & ( ! (N^5 (bus & (!b))))
```

where `b` is instantiated to `pos1`, ..., `pos14`, and `N^k`, for k a natural number, is the nested application of the *near* operator (also called *dilation* in the context of computational imaging). The formula dilates `b` by three pixels, and avoids points too close to *other* buses, in order to minimise possible overlaps. In the same figure, in red, we colour positions that are close to a bus stop. This uses the formula `close(bus,stop)`, where `close` is defined as follows:

```
Let close(a,b) = a & (N^30(b))
```

Thus, the formula intersects the points where the bus has passed with the points reachable from a stop by 30 pixels.

Implausible points Data points that are not positioned on a street are implausible. This is described by the formula:

```
Let busOutOfStreet =
    bus & (! (bus S street))
```

The formula characterises points that are part of any of the regions denoting bus positions, and are not surrounded by a street. Points satisfying this formula (thus, not plausible) are repainted in violet to produce the spatial model-checking result shown in Figure 6.

Possible points Diverted bus positions are represented by the formula:

```
Let divertedBus = bus S smallStreet
```

and are then repainted in red to produce the result shown in Figure 6.

Problematic points Our spatial logic is expressive enough to define properties related to the order of positions of the same bus at different times on a given road. We analyse the bus positions. For each position, we detect its immediately neighbouring positions, in order to check that these correspond to a preceding and following position, respectively. If this is not the case, the position is misplaced, even though it may still be on the main street. In Figure 7, such misplaced bus positions are painted in red. The formal specification of this property is complicated by the fact that the underlying graph of an image is not directed, thus it is not completely straightforward to specify precedence relations between points. The most important step is the definition of predicate `consecutivePos(p1,p2)`, given below. The definition uses the *reachability* predicate $a\ \mathcal{R}\ b$, written as `reach(a,b)`.

```
 Let consecutivePos(p1,p2) =  p1 S
    reach((streetPortion(p1) | mainStreet | streetPortion(p2)) &
                (!(streetPortion(bus & (!(p1 | p2))))),
           streetPortion(p2))
```

The definition of `consecutivePos` uses the predicate `streetPortion` to identify the region of points in `p1`, surrounded by the area from which `streetPortion(p2)` can be reached, passing by the main street, including the areas surrounding `p1` and `p2`, avoiding the areas surrounding other buses. Formula `streetPortion(p)` is defined in order to denote an area at least as wide as the main street, depending upon the map considered. By the definition of `streetPortion`, no next bus positions can be reached following only points belonging to the main street. Using `consecutivePos`, predicate `wrongOrderPos` is defined as follows:

```
 Let wrongOrderPos(p1,p2,p3) =
    (p2 S mainStreet) &
        (!(consecutivePos(p2,p1) &
           consecutivePos(p2,p3)))
```

Given three positions `p1`, `p2`, `p3`, position `p2` is selected only if it is not strictly between `p1` and `p3`. The three positions are instantiated to all the strictly consecutive triplets between `pos1` and `pos14` in order to identify out-of-order positions.

Although the above examples show that purely spatial logic can be used to deal with some (albeit limited) spatio-temporal properties, in general it is more convenient (and expressive) to use a dedicated spatio-temporal logic. How this works is illustrated in the next section.

5 Spatio-temporal model checking

Spatio-temporal logics may be defined by permitting mutually recursive nesting of spatial and temporal operators. Several combinations are possible, depending on the chosen spatial and temporal fragments, and the permitted combinations of the two. A great deal of possibilities are explored in [31], for spatial logics based on topological spaces. One such structure was explored in [13, 26], in the setting of closure spaces. The current implementation is provided by the open source spatio-temporal model checker[4] `topochecker`, that can verify formulas written in the *spatio-temporal logic of closure spaces* STLCS. Such logical language enhances SLCS with temporal operators, in the spirit of the branching-time temporal logic CTL (Computation Tree Logic) [17].

In this section we provide a lightweight informal introduction to spatio-temporal model checking; we refer the interested reader to [13] for further technical details. Like SLCS, this spatio-temporal logic is developed in the setting of closure spaces. In addition to the already discussed spatial operators, STLCS features the CTL path quantifiers `A` ("for all paths") and `E` ("there exists a path"). As in CTL, such quantifiers must necessarily be followed by one of the path-specific temporal operators $\mathtt{X}\Phi$ ("next"), $\mathtt{F}\Phi$ ("eventually"), $\mathtt{G}\Phi$ ("globally"), $\Phi_1 \mathtt{U} \Phi_2$ ("until"), where Φ, Φ_1 and Φ_2 are STLCS formulas. For a further introduction to and more details on CTL and CTL model checking, the reader may consult [3].

A model $\mathcal{M}$ of the STLCS logic is composed of a Kripke structure $(S, \mathcal{T})$ and a closure space $(X, \mathcal{C})$. More precisely: S is a non-empty set of *states*; $\mathcal{T}$ is an *accessibility relation* on states, which is required to be *left-total*, that is, for each s in S there is s' in S with $(s, s') \in \mathcal{T}$; X is a set of points; $\mathcal{C}$ is a closure operator (see Section 2). Every state s has an associated valuation $\mathcal{V}_s$, making $((X, \mathcal{C}), \mathcal{V}_s)$ a *closure model* according to Definition 6 of [14]. An equivalent interpretation is that the valuation function has type $S \times X \to 2^P$, where P is the set of atomic propositions, thus, the valuation of atomic propositions depends both on states and points of the space. The evaluation contexts are of the form $\mathcal{M}, s, x \models \Phi$, where Φ is a STLCS formula, s is a state of a Kripke structure, and x is a point in space X. In both notations, the intuition is that there is a set of possible worlds, i.e. the states in S, and a spatial structure represented by a closure space. At each possible world there is a different valuation of atomic propositions, inducing a different "snapshot" of the spatial situation which "evolves" over time. This is made clear along a temporal path. A path in the Kripke structure denotes a sequence of snapshots of the considered space, indexed by instants of time; see Figure 8 for a pictorial illustration.

Let us proceed with a few examples. Consider the STLCS formula `EG (green S blue)`. This formula is satisfied in a point x in the graph, associated to the initial state s_0, if there exists a (possible) evolution of the system, starting from s_0, in which point x is always, i.e. in every state in the path, green and surrounded by blue ('*green* $\mathcal{S}$ *blue*' in the terminology of Section 2). Note that the model-checker will return (or colour) *all* the points x that satisfy the formula. A further, nested, example is the STLCS formula `E F (green S (A X blue))`. This formula is satisfied in a point x in the graph associated to the initial state s_0, if there is a

[4] See `http://topochecker.isti.cnr.it/`, and `https://github.com/vincenzoml/topochecker`; an earlier prototype is available at `https://github.com/cherosene/ctl_logic`.

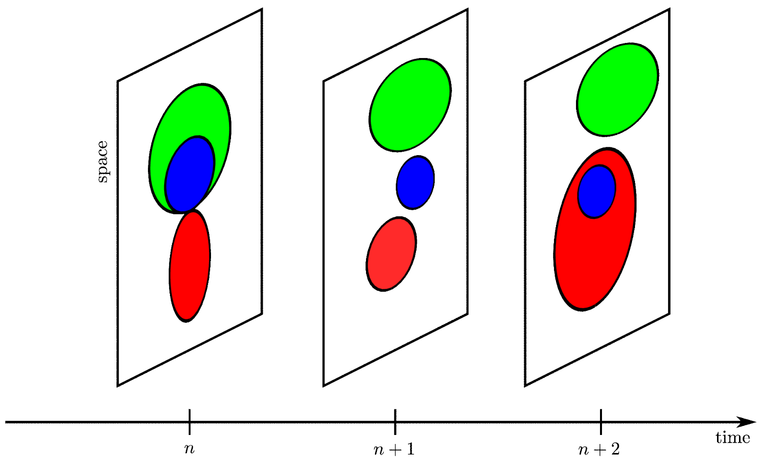


Fig. 8: In spatio-temporal models, a temporal path is a sequence of *snapshots* that are induced by the time-dependent valuation of the atomic propositions.

(possible) evolution of the system, starting from s_0, in which point x is eventually green and surrounded by points y that, for every possible evolution of the system from then on, will be blue in the next step.

We briefly recall the formal semantics of STLCS that was first presented in [13].

Definition 2 Satisfaction is defined in a STLCS model $\mathcal{M} = ((X, \mathcal{C}), (S, \mathcal{R}), \mathcal{V}_{s\in S})$ at point $x \in X$ and state $s \in S$ as follows:

$$\begin{array}{lll}
\mathcal{M}, x, s \models \top & & \\
\mathcal{M}, x, s \models p & \iff & x \in \mathcal{V}_s(p) \\
\mathcal{M}, x, s \models \neg\Phi & \iff & \mathcal{M}, x, s \not\models \Phi \\
\mathcal{M}, x, s \models \Phi \vee \Psi & \iff & \mathcal{M}, x, s \models \Phi \text{ or } \mathcal{M}, x, s \models \Psi \\
\mathcal{M}, x, s \models \mathcal{N}\Phi & \iff & x \in \mathcal{C}(\{y \in X | \mathcal{M}, y, s \models \Phi\}) \\
\mathcal{M}, x, s \models \Phi \,\mathcal{S}\, \Psi & \iff & \exists A \subseteq X. x \in A \wedge \forall y \in A. \mathcal{M}, y, s \models \Phi \wedge \\
& & \quad \wedge \forall z \in \mathcal{B}^+(A). \mathcal{M}, z, s \models \Psi \\
\mathcal{M}, x, s \models \mathtt{A}\,\varphi & \iff & \forall \sigma \in \mathcal{P}_s.\, \mathcal{M}, x, \sigma \models \varphi \\
\mathcal{M}, x, s \models \mathtt{E}\,\varphi & \iff & \exists \sigma \in \mathcal{P}_s.\, \mathcal{M}, x, \sigma \models \varphi \\
& & \\
\mathcal{M}, x, \sigma \models \mathcal{X}\Phi & \iff & \mathcal{M}, x, \sigma(1) \models \Phi \\
\mathcal{M}, x, \sigma \models \Phi\,\mathcal{U}\,\Psi & \iff & \exists n.\, \mathcal{M}, x, \sigma(n) \models \Psi \text{ and } \forall n' \in [0, n). \mathcal{M}, x, \sigma(n') \models \Phi
\end{array}$$

In this definition $\mathcal{B}^+$ denotes the *closure boundary*, as in Def. 1, and σ denotes a path, starting at index 0, that is, $\sigma(n)$ denotes the $(n{+}1)$-th element of σ. A path in the Kripke structure is a sequence of *spatial models* (in the sense of [14]) indexed by instants of time and $\mathcal{P}_s$ denotes the set of paths starting from state s.

Spatio-temporal model checking is performed using a variant of the classical CTL labelling algorithm [17,3], augmented with the algorithm in [14] for the spatial fragment. In the implementation the temporal component of a model is a Kripke structure, represented by a graph. The latter uses the plain text graph description language `dot` for graph representation[5]. The underlying spatial structure is fixed for all states. It can be either a graph, or a regular grid implicitly defined when the valuation of atomic propositions is given by digital images (as done in this paper). In the first case, the graph is defined using the `dot` notation, and the valuation of atomic propositions, depending upon states and points of the space, is provided separately, using a *comma-separated values* file. When digital images are used, for each state, a digital image is provided for each state in the Kripke structure. Images must have the same size, and the corresponding grid is taken as the reference closure space $(X, \mathcal{C})$. (Recall that grids are graphs, and therefore instances of closure spaces, see [14] for further details.) Colours of the points of the picture for each state s determine the valuation function $\mathcal{V}_s$. The model checker receives as input also a list of logic formulas to be verified, with a colour labelling each formula.The output of the model checking algorithm is the model $((X, \mathcal{C}), (S, R), \mathcal{V}'_s)$ where the closure space $(X, \mathcal{C})$ and the Kripke structure (S, R) are the same as the input ones, whereas $\mathcal{V}'_s(x) = \mathcal{V}_s(x)$ if x does not satisfy any formula in the list, otherwise it is the colour assigned in the list to the formula it satisfies (when formulas overlap in image-based models, the colour of the last formula is used). Using spatial models based on images makes interactive exploration of the model checking results possible. More precisely, one can use the *graphviz* toolkit to visualize the graph corresponding to the Kripke frame given as input, in such a way that clicking on a state visualizes the image associated to it in the model checker output. This feature proved to be useful in understanding the results of our case study (see Section 7). Furthermore, with reference to the same section, we mention that in the implementation of the model checker, states are numbered, and the output of the model checking can be accessed by these numbers (more precisely, in the implementation, to each state s, a coloured image is associated in the output of the model checker, corresponding to the projection $\mathcal{V}_s$ of the valuation function).

Several derived operators can be defined starting from the basic ones whose semantics is given in Definition 2. Among these, we recall the path operators F ("eventually") and G ("globally"). For example we have $\mathcal{M}, x, s \models \mathtt{AF}\phi$ whenever for all paths σ in $\mathcal{P}_s$ there is i with $\mathcal{M}, x, \sigma(i) \models \phi$; similarly, for the `AG` variant, replace "there is i", with "for all i".

6 Bus operational issues: analysing headway

An instance of a short headway problem is illustrated in Figure 9 using a series of successive "satellite view" images of the bus data (from the top-right corner to the bottom-left corner of the picture).

In the literature various proposals exist that aim at dynamically maintaining an acceptable headway in bus systems (see for example some recent work [29, 42, 20, 19] and references therein). A particular class of such proposals is based on bus holding strategies. These are self-adaptive strategies where buses are put on hold for a short period of time. This can be at predefined control points, such as at the end-points of a route, or this can be at the bus stops themselves. The

[5] Further information on the `dot` notation and the *graphviz* toolkit can be found at http://www.graphviz.org/Documentation.php.

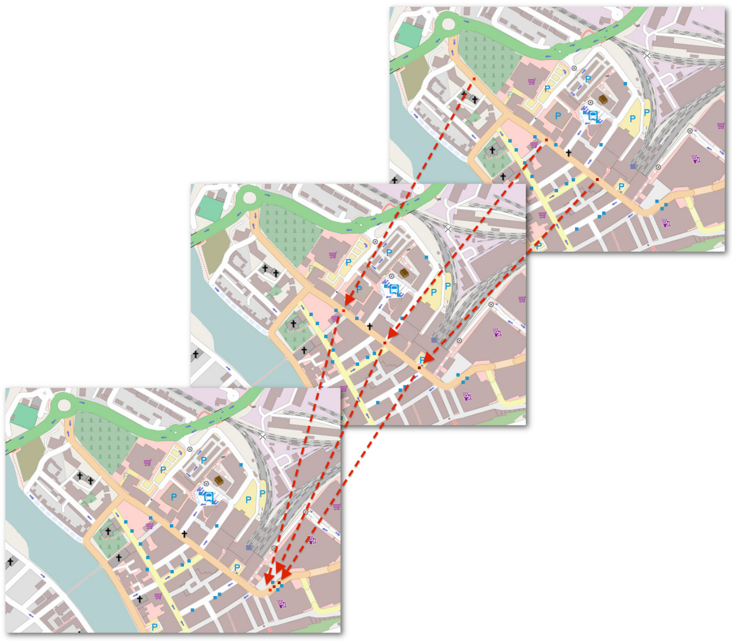


Fig. 9: Because of delays caused by boarding passengers the headway between buses is successively eroded over time until the buses are essentially working as a conglomerate.

waiting time is usually established by a function that depends on the backward headway, that is, the headway between the current bus and the bus behind, or the forward headway, namely, the headway between the current bus and the bus in front (see [42]), or both. Perhaps the most obvious measure of headway is *chainage*, the *distance by road* between two vehicles. However, because buses change speed on different parts of the route, this is not the best metric to use. One usually wishes to address time-related problems with the service (passengers waiting for too long a time between buses; and there not being enough time between buses for passengers to arrive at a bus stop). For this reason, the bus holding strategies discussed above rather focus on the *temporal separation* between buses. In order to express such notions of headway, one should take into account both spatial and temporal aspects; their interplay may have rather subtle aspects to consider, and require careful formalisation.

Using spatio-temporal logic, clumping of buses can be detected, both in a system trace (e.g., a GPS trace obtained at run-time), and in more complex *branching* models, that are used in Section 7. Consider a single bus route, served by k buses. At each instant of time, the state of the system is completely described by a tuple of k GPS positions; therefore, a system trace is a finite sequence of such tuples. As already discussed, there may be several different ways to formalise the notion of clumping. We describe two possible variants, with the idea in mind to check for clumpings that happen at a bus stop (clumping could happen anywhere, but

```
// buses are shades of red:
Let bus1 = [red == 155] & [green == 0] & [blue == 0];
Let bus2 = [red == 188] & [green == 0] & [blue == 0];
Let bus3 = [red == 221] & [green == 0] & [blue == 0];
Let bus = bus1 | bus2 | bus3;

//bus stop is dark blue:
Let busStop = [red == 55] & [green == 55] & [blue == 255];

Let close(x) = N^7 x;

Let busAtStop(x) = busStop & close(x);

Let busAfterBus(x) = busAtStop(x) & E X busAtStop(bus & !x);

Let closeToOtherBus(x) = (x & close(bus & !x));
Let conglomerate = busStop & close(closeToOtherBus(bus1)
                   | closeToOtherBus(bus2) | closeToOtherBus(bus3));

Let timeConglomerate = busAfterBus(bus1) | busAfterBus(bus2) | busAfterBus(bus3);

Check "0x00FF00" (E F timeConglomerate); // stops with conglomerate turned green
```

Fig. 10: Spatio-temporal formulas for conglomerates

it is significant from an user point of view only at bus stops). Figure 10 contains the input of a model-checking session using three buses. Formulas bus1, bus2, bus3 characterise different buses on the same route in different shades of red and busStop denotes a bus stop indicated by a dark blue square. In this example, shades of colours (of red in this case) are used to distinguish the different buses serving the same route, so that each bus has a specific shade of the colour through time. The formulas that we explain below are true at points of a bus stop whenever clumping (i.e. formation of a conglomerate) is happening at that particular bus stop.

1. A *spatial* conglomerate happens when two buses serving the same route, at some point in time, are spatially close to each other, and also close to a bus stop. This event is described by the formula conglomerate. Points satisfying this formula are those that are close to a bus, which is in turn close to another bus. Formula closeToOtherBus, which is parametrised by the chosen bus, is responsible for checking that a bus is close to another one. The notion of "closeness" is defined by formula close, using nested applications of the basic closure operator of the logic (the number of nested applications has been determined by trial and error in this example, but it can easily be related to real-world distances).
2. A *spatio-temporal* conglomerate happens when two buses serving the same route pass by the same stop in a short amount of time. This case is subtler than the previous one, as it does not necessary imply that the (spatial) distance between two subsequent buses becomes too small. One way to describe such event is by the formula timeConglomerate, which features a combination of spatial operators (used to detect that a bus is close to a stop) and temporal operators (used to identify the spatio-temporal conglomerate). For instance, consider the formula busAfterBus1. This formula is true on points that are:

i) part of a bus stop, and close to `bus1`, because `busAtStop` must be true for `bus1`; ii) such that, in one[6] time step, these will be part of a bus stop, and close to either `bus2` or `bus3`. Note that the use of spatial and temporal connectives in the same formula permits one to refer to the colour of points at a specific time, and at subsequent time instants. However, we remark that complexity of model checking spatio-temporal conglomerates in this way is proportional to the number of considered buses. In Section 7 we will thus use a slightly more elaborated notion of spatio-temporal clumping, which does not require to instantiate a sub-formula for each bus.

Once established what is the kind of clumping one is interested in, one may use temporal operators to detect points where, for example, clumping will happen at some point in the future. Figure 11 is obtained with `topochecker`, starting from the positions of three buses serving the same route. Figures 11a-11e are obtained by mapping bus coordinates over a base map. Buses are represented by squares of different shades of red. The dark blue square is a bus stop. Figure 11f shows the output of the model checker in the initial state when checking the formula `E F timeConglomerate`. Indeed, Figure 11f is the same as Figure 11a, except for the colour of one bus stop, whose points are coloured green as the result of the model checking procedure indicating that clumping happens at that stop, eventually in the future.

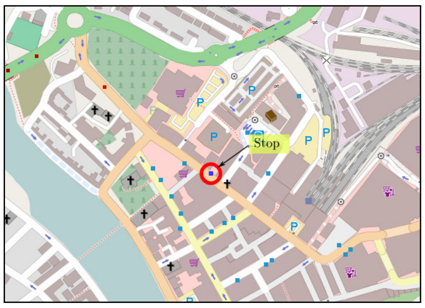
As the model-checker is a simple prototype, we do not provide accurate performance information. We just remark that execution time is in the order of a few seconds on a standard laptop for this example, in which over one million points (approximate size of the image) are examined several times (proportional to the number of sub-formulas of the formula to be verified).

7 Analysing the effect of bus holding strategies for operational issues

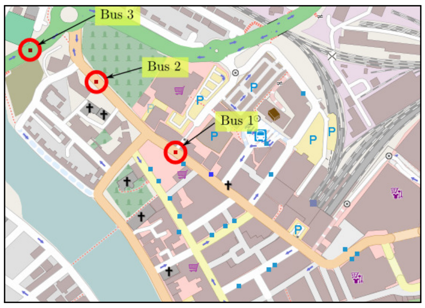
Several bus holding strategies suggested in the literature issue wait instructions to buses to rebalance the headway between them. The additional waiting period usually depends on the forward or backward headway (or both) of a bus at one or more control points selected along the route (see for example [29,42]). Other strategies suggest a maximal speed limit that is continuously updated and communicated to the driver based on GPS positioning data of preceding and following buses. Although the performance of these strategies have been studied in detail both from a theoretical point of view and by means of simulations, it is much harder to foresee whether a particular strategy might work in a real situation. Some tests in real cities have been performed as well (see for example [28]) but these may be very costly and time consuming to perform. In the following we describe a method by which spatio-temporal model checking could contribute to analyse the effects of bus holding strategies in real cities by analysing models obtained from a transformation of linear traces of actual AVL data of buses on a given route.

We consider a simplified scenario of a richly-instrumented real-time-informed system where data cleaning has been applied as described in Section 4 to result in

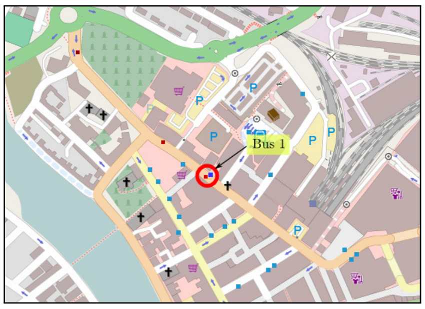
[6] More than one time step can be required. This can be achieved by repeated nesting of applications of `E X` operators. We did not do so for the sake of clarity in Figure 11.



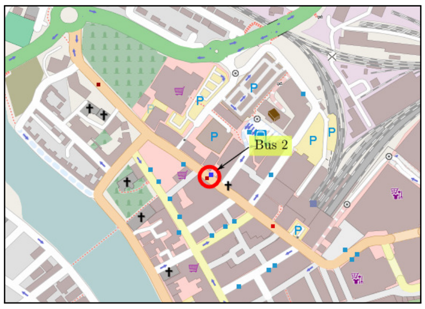
(a) Initial state



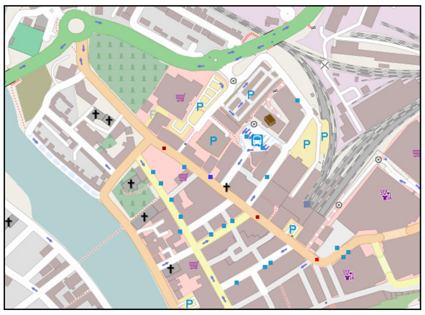
(b) Second state



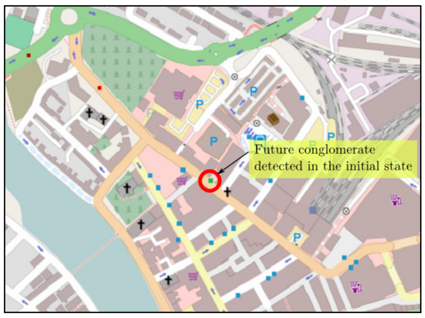
(c) Bus 1 passes by the stop



(d) Bus 2 passes by the stop



(e) Final state



(f) Result from the model-checker. Points of the initial state that will be involved in a time conglomerate are coloured

Fig. 11: Spatio-temporal conglomerate.

a plausible set of observations of bus positions. In this scenario, either driven by the central authority of the bus system, or by an autonomous, adaptive response, buses could intervene when short headway problems are detected by waiting for a predefined amount of time to lengthen the headway. This intervention may not necessarily solve the problem, but it improves it, and it is always possible to let a bus wait again later if necessary. Differently from the strategies in the literature, we do not let the waiting time depend on the headway of preceding or following buses. However, multiple wait instructions can be issued, which is similar to waiting for a longer time interval. Because the wait instructions are issued in response to a detection of the shortening of the forward headway, the strategy is also (self)-adaptive to some extent. The simplified approach we assume is quite permissive on when and where wait instructions are sent. As such, it only serves the purpose of exemplifying how to use spatio-temporal model checking for assessing the impact of a mitigation strategy; studying more elaborate holding strategies is left to future research. Note, however, that our analysis methodology for policies does not depend on the chosen policy, and our simplified strategy is used merely for exemplification purposes. More complex strategies could be checked as well. The complexity of the model transformation applied to the traces depends upon the features that are observed (e.g., distance between buses, headway, number of passengers, delay, etc.).

Note that the spatio-temporal model checker can be used both to detect clumping in a system trace, as we have seen, or to analyse a *branching* model where at each state, non-deterministically, there may be several possible steps to different future states. Such non-deterministic models represent in a concise way a great number of possible system behaviours or scenarios, depending on the choices that may be made at each execution step. The possibility of issuing wait instructions to specific buses, or not doing so, introduces non-deterministic choice points.

We designed and implemented an experiment, demonstrating how, starting from real-world GPS traces of a bus network, one can generate a branching spatio-temporal model that mimics the effect of possibly non-deterministic policies aimed at modifying the system behaviour, analyse complex spatio-temporal properties by means of the spatio-temporal model checker, and interactively inspect the results, as we explained in Section 5, by looking at the behaviour of the system, represented as a tree of possible states, and at maps with super-imposed bus positions and bus stops, coloured according to the results of model checking several properties. The model generator that we implemented, and the model checker, constitute a tool-chain which is able to accept as input a set of GPS traces and produce as output a spatio-temporal model, coloured according to the Boolean valuation of user-defined spatio-temporal formulas. We used the tool-chain on real data and examined the results of some relevant formulas.

7.1 Model generator

We developed a model generator[7] taking as input a list of GPS positions (in the form of a *comma separated values* file) of several buses of the same route. Each item in the list consists of a bus fleet number, a pair of GPS coordinates (expressed in decimal degrees), and a time-stamp (year, month, day, hour, minute, second). All the buses are operating on the same route. The GPS traces of the buses are not assumed to be synchronous, that is, there is no pre-established time interval at which the GPS positions of all buses on the route are collected, but rather, the time-stamp of each bus position is independent from the others. Furthermore, the model generator takes as input a list of bus stops, with their GPS coordinates, a digital image of a portion of a world map (used as background

[7] The source code, written in the programming language OCaml, is available as a free and open source software in the source code repository of `topochecker`.

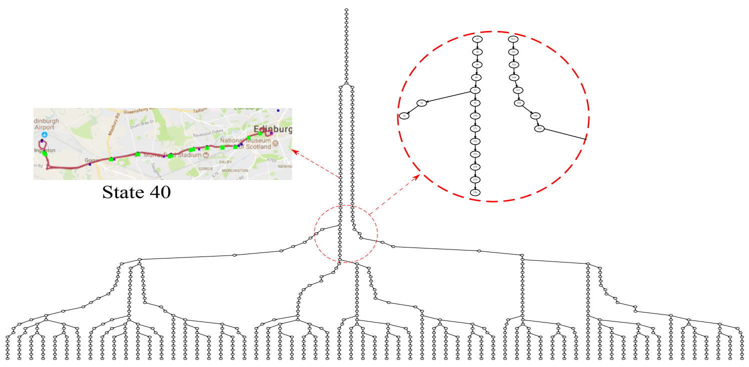
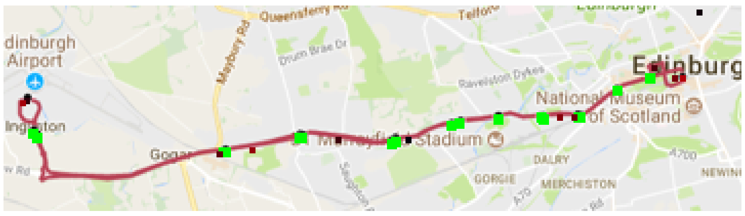


Fig. 12: Branching spatio-temporal model generated by a bus holding strategy. State 40 is detailed using the corresponding image; this state is analysed in one of our examples in Section 7.

for the output, as described below), a textual description of the GPS coordinates of its bounding box, and a set of parameters that affect model generation, namely the time step length (in minutes), the start and end of the considered time window (in minutes), the number of time steps during which a bus should rest in response to a "wait" instruction, the maximum delay allowed for a bus (in minutes), and three parameters related to clumping detection, namely a spatial and a temporal distance, and a clumping duration expressed in time steps.

The output of the model generator is a synchronous branching spatio-temporal model (see Figure 12) suitable for being loaded by `topochecker`. The model is synchronous in the sense that a system state is computed for each "time step", that is, a user-defined time interval. The model is branching, since at each point in time, a "clumping avoidance strategy" (described below) may instruct buses to wait for a user-defined amount of time. In that case, choice points are introduced in the temporal flow of the model, as waiting is considered a possible choice affecting the possible future behaviours of the system. Model checking is thus used to exhaustively explore all the possible behaviours.

The generated model consists of a tree representing all the system states, and, for each state, a digital image (an example is reported in Figure 13) consisting of the background (portion of world map) described above, with superimposed the computed positions of all buses on the route, and the bus stops. Each bus position is colour-coded by changing the RGB coordinates of its colour at each state as follows. The blue component is fixed to one of three values which are not present in any point in the background map (this permits one to pinpoint a bus superimposed to the map). The three possible values of blue are used to indicate the direction of the bus along the route; there are two values for the two possible end-to-end directions, plus an "unknown" value, needed since in our experiment, the direction of each bus is not included in the AVL data, therefore it needs to be detected algorithmically, which is not always possible (e.g. when there is only one data point, at the beginning or at the end of the model generation process). The intensity of the red component is proportional to the bus identifier. The intensity of the green component is proportional to the delay of the given bus in the generated state. All buses start with no delay. Bus stops are painted in green.



Map data copyright 2017 Google

Fig. 13: One state in the generated model. Buses are colour-coded according to their direction, fleet number, and current delay. Bus stops are painted in green.

Remark 1 We note that, although data about the direction of buses serving each stop is available, and the model generator can encode bus stops of each direction differently, we did not use this feature in our experiment, as the bus stops corresponding to opposite directions are very close to each other in the considered route, and using data about their direction would generate too much noise in this particular case. Therefore, bus stops of the same location, related to opposite directions, are intentionally painted in the same colour in our figures and cannot be distinguished by the model checker.

7.2 Clumping avoidance strategy

We have chosen a simple behavioural strategy for clumping mitigation that detects clumping at each instant of time using only past information about bus positions, and instruct buses to wait – up to a maximum delay – when clumping is detected, lasting a certain amount of time. Such a strategy, which could be easily implemented even without centralised control, has obvious drawbacks (e.g., the whole network can be delayed due to a single local phenomenon), but it serves well our purpose to check feasibility of spatio-temporal analysis based on images, using real-world data and a realistic behavioural model generation.

Getting into more detail, for each bus, the internal state of the model generator includes the current delay, and the past positions occupied by the bus at each time step, since the first time step in which information about that specific bus was found in its GPS trace. For time steps in which more than one GPS position is found in the input data, positions are averaged, whereas for time steps in which no position is available, the previous position is used. Clumping is defined by three parameters, namely a *temporal distance* t, expressed in time steps, a *spatial distance* s, expressed in meters, and a *clumping duration* d, expressed in time steps. We then define the asymmetric clumping relation $c(b_1, b_2)$ where b_1 and b_2 are bus identifiers. Relation $c(b_1, b_2)$ holds whenever b_1, following b_2, is at a bus stop – therefore able to wait for some time – and, for each time step x in the last d steps, there is a time step y between $x - t$ and x in which the spatial distance between the position of b_1 at x and the position of b_2 at y is less than or equal to s. The role of the parameter t is to define a maximum temporal interval between two buses to consider them in a situation of clumping. If the situation is as described above, this means b_1 passed by "roughly the same" (parameter s) points as b_2 in a "too short amount of time" (parameter t), for "too long" (parameter d). Therefore b_1 should be delayed. For instance, taking $d = 2$, $t = 5$ and $s = 500$ meters, clumping occurs for b_1 if b_1 passes by a point x (+/- 500 meters, i.e. s) within 5 time steps (i.e. t) after b_2 passed by x, and this happens again in the next time step for a point x' where b_2 passed by and b_1 reached x' within 5 time steps after b_2. If the current delay of b_1, plus the user-specified waiting time, is less than or equal to the maximum allowed delay time for a bus (which is also user-defined), then a choice point is generated, corresponding to the possibility of letting b_1 wait; the future behaviour is thus divided into two branches. On one branch, model generation proceeds as if no clumping has been detected. On the other branch, all the future positions of b_1 are delayed by the specified wait time. Note that there may be a bus b_3 following b_1, that becomes in turn too close to b_1 because b_1 is waiting. This may indeed be detected as clumping in some subsequent step of the generation process, and produce another branch in the generated model.

The model generation that we implemented is rather primitive in nature. In particular it does not account for guessing intermediate positions of a bus when no position is available in one time interval, and does not take into account the fact that a bus may be delayed due to external conditions (e.g. sudden, unpredictable traffic jams) that would therefore affect all the future positions of a bus and would require, e.g., a discounting factor in computing future positions after a bus rests for a given time. Such considerations, together with experimentation with more complex strategies (see e.g., [19]) can be easily accounted for in our methodology, by changing the clumping detection function and the computation of derived traces. However, the simple method that we implemented is sufficient for interactive experimentation for model checking purposes, and demonstrates usability and feasibility of a tool-chain comprising branching model generation and spatio-temporal model checking in the context of *smart transportation.*

7.3 Experiment set-up, model size and execution time

We used as input the GPS traces of 12 buses over route 100 (Airport-City Centre) in Edinburgh, U.K., kindly provided by Lothian Buses. We experimented with a trace length covering one hour and a half (drawn from several days of original GPS traces), which in our tests has proved sufficient to provide interesting behaviour without giving rise to unmanageable model sizes. We remark that it would not be very interesting to employ much longer trace lengths; it is expected that any reasonable bus holding strategy would "reset" the maximum delay after some period of time, to permit the strategy to take place again. Otherwise, the "maximum delay" parameter of the model would quickly prevent any further response to clumping after all buses have reached the maximum value. In the model we generate, there are buses that reach the maximum delay within the given time frame, therefore longer traces would not be significant in evaluating properties of bus holding strategies such as the one we investigate.

Although we did not compute accurate statistics on our data, we can provide some estimates on model sizes. Approximately, one hour and a half of input data are covered by 2000 input GPS positions. The generated branching model (see Figure 12) consists of about 800 states. A digital image (pixel size: 340x100) is associated to each state. The on-disk size of the generated model (including all files, suitable for analysis by `topochecker`) is about 80MB. All test were executed on the same machine (see Section 7.4). The time for model generation is in the order of 10 seconds. As we shall see, analysis via model checking, depending on the considered formula, ranges between a few seconds and a few minutes resulting in a rapid work-flow for interactive refinement of the logic formulas that are used. Such low numbers indicate that larger models (e.g. using higher-resolution images, or more detailed input traces) would be tractable. However, as we already explained, the considered data size is appropriate for the task that we illustrate, and in our experiments there was no need for heavier workloads.

The model generation parameters (see Section 7.1) were instantiated as follows. Time step length was set to 1 minute; the considered time window starts at 7 a.m. and ends at 8.30 a.m.; the number of time steps during which a bus should rest in response to a wait instruction was set to 2 time steps; the maximum delay allowed for a bus was set to 6 minutes. The parameters for clumping detection were set as follows: spatial distance set to 500 meters; temporal distance set to 5 time steps; clumping duration set to 2 time steps. As the route is served on average by one bus every 10 minutes, such parameters seem realistic. We experimented with variations of the chosen parameters without observing noticeable changes of model size and computation times.

7.4 Spatio-temporal properties

The spatio-temporal model obtained as described above contains a lot of information, relating bus positions to several features, e.g. individual bus delay, spatial aspects (such as the position of bus stops, or the distance between buses) and temporal aspects. Looking at such highly structured information may be a hard task; although numerical indicators may be derived, these do not capture the complexity of the observed phenomena. Our spatio-temporal logical language is used as a *query language*, in the spirit of queries over structured data in traditional relational databases, in order to extract complex structured information from the data of the generated model.

In the remainder of this section, we illustrate some relevant queries for the *smart buses* scenario, explain how they are built, and show the model checker output. Such output consists of an image for each state s of the model, coloured according to the truth value of each formula at each point of the model at state s. We just show the image associated, in the model checker output, to the initial state, as this is sufficient to get a glimpse on how such formulas operate; however, part of the effectiveness of using `topochecker` in this case study is the possibility of interactive exploration of the results (as we mentioned in Section 5), by using *graphviz* to visualize the Kripke model, and open the associated image in the model checker output, when clicking on a state. For each considered formula, we also provide a rough estimate of execution times; these are based on a desktop machine equipped with an *intel core i7* processor and 8 gigabytes of central memory. CPU power directly (linearly) affects execution time, since the spatio-temporal model-checking algorithm is linear in the size of the model. The algorithm is a global in-memory model checker, therefore RAM size only affects feasibility of the analysis, since the model must fit into memory to be analysed; we note that in our

```
Let centreBus = [blue == 10];
Let airportBus = [blue == 15];
Let unknownBus = [blue == 13];

Let bus = centreBus | airportBus | unknownBus;
Let bus1 = bus & [red == 0];
Let bus2 = bus & [red == 15];
Let bus3 = bus & [red == 30];
Let bus4 = bus & [red == 45];
Let bus5 = bus & [red == 60];
Let bus6 = bus & [red == 75];
Let bus7 = bus & [red == 90];
Let bus8 = bus & [red == 105];
Let bus9 = bus & [red == 120];
Let bus10 = bus & [red == 135];
Let bus11 = bus & [red == 150];
Let bus12 = bus & [red == 165];

Let delayed = [green > 0];
Let delayed1min = [green == 5];
Let delayed2mins = [green == 10];
Let delayed3mins = [green == 15];
Let delayed4mins = [green == 20];
Let delayed5mins = [green == 25];
Let delayed6mins = [green == 30];

Let busStop = [red == 0] & [green == 255] & [blue == 0];
```

Fig. 14: Abbreviations for relevant image features based on the colour coding of the model generator.

experiment, the model size is just about 80MB, which is two orders of magnitude smaller than the maximum available at the machine in use.

First of all, we need to set up some abbreviations for formulas that have a concrete meaning in our scenario, namely bus stops, buses, direction, and delays, based on the colour coding used by the model generator. Such abbreviations are reported in Figure 14, and their meaning should be self-explanatory. Note that formulas `centreBus`, `airportBus`, and `unknownBus` represent directions (going to the centre, going to the airport, or unknown). The direction of a bus is detected algorithmically in the model generator, and can therefore be unknown e.g. when there is only one data point, or when there are zero (both things only happen at the beginning and end of the given AVL trace).

Furthermore, we need to introduce some general-purpose definitions of parameterised formulas (that is, derived operators), reported in Figure 15, and formally introduced in Section 2 and [15]. Let us recall the meaning of the derived operators with the notation used in the tool. Formula `touch(x,y)`, based on the inner formula `reach(x,y)`, identifies all the points laying in an area, say A, such that each point in A directly touches either one point satisfying y or another point in A, and all points in A satisfy x. Formula `everywhere(x)` is a global assertion operating on connected spaces; it denotes all points of the space, when all points satisfy x, and no point otherwise. Formula `somewhere(x)` is a “global” assertion (on connected spaces) in turn; it either denotes all points of the space, when there exists at least one point satisfying x, or no point, if there is no point satisfying x.

```
Let reach(x,y) = !((!y) S (!x));
Let touch(x,y) = x & reach(x|y,y);
Let everywhere(x) = x S FF;
Let somewhere(x) = !(everywhere(!x));
```

Fig. 15: Some general-purpose parameterised formulas.

First, one can check whether the considered trace length (one hour and a half) is enough to capture the interesting dynamics of the chosen bus holding strategy. In the case of our simple strategy, we assume that if buses reach their maximum waiting time, they should soon be "reset" (e.g., by allocating a waiting time in the final destination, that can be adjusted depending on the situation). Long traces become less significant if some buses reach their maximum delay in the generated model. This is checked via the formula

```
somewhere(E F (bus & delayed6mins));
```

The tool `topochecker` takes 30 seconds to verify this formula, and colours in red all the points of each state (but note that it is sufficient to look at the first state), demonstrating that there are buses that reach the maximum delay time. Note that there are, necessarily, paths where buses do not reach the maximum delay (as wait instructions generate choice points in the model, therefore they are "optional").

One problem of the considered strategy is that it may slow down the whole network. The considered bus route has an expected frequency of one bus every 10 minutes. We will now use the model checker to verify if there are bus stops where no buses pass by for 15 minutes, which would be a noticeable delay. This will be done by model checking the generated model, considering both the case in which no bus is delayed (therefore, the problem is intrinsic to the considered data), and the case where delay is permitted (therefore, additional problems found are to be related to the chosen behavioural policy).

In order to consider only system states with / without added delay, we first define the formulas `noDelay` and `delay`:

```
Let noDelay = everywhere(!(bus & delayed));
Let delay = somewhere(bus & delayed);
```

Next, we define two formulas related to the presence of buses at stops. Formula `busAtStop(dir)` identifies points belonging to a stop where there is a bus travelling according to the specified direction. Formula `emptyStop(dir)` identifies points belonging to a bus without any bus travelling in the given direction (note that, by Remark 1, we do not take into account the direction of each stop).

```
Let busAtStop(dir) = touch(busStop,dir);
Let emptyStop(dir) = busStop & (!touch(busStop,dir));
```

The next formula is parameterised with a parameter *del*, which will be clarified later and is related to the delay of buses, a direction, and a continuation *cont*. It is used to identify a bus stop that is empty (with respect to buses passing in the specified direction), satisfies *d*, and in at least one possible next time step, also satisfies *cont*.

```
Let noBusesStep(del,dir,cont) =
   emptyStop(dir) & del & (E X cont);
```

Passing a continuation is used to simplify formula `noBusesFifteenStepsDir` described below. Such formula is still parameterised over *del* and *dir*, and computes the points belonging to a bus stop which is not served by any bus in the given direction for 15 steps, and satisfies *del* at each step. Additionally, we check that the last step is not a final state (which implies that also intermediate steps are not final). Not including final states is needed as those states are the points where the path is artificially truncated by the end of the model generation process. Therefore such states exhibit an artificial infinite loop ("deadlock") that should not be taken into account, for example because a bus stop that is empty in a deadlock state will stay empty forever. Such deadlocks are the result of using finite models (such as the model we generate) when considered as *Kripke structures* in model checking. The model checker `topochecker` defines a special predicate, named `deadlock`, to denote points of such states.

```
Let noBusesFifteenStepsDir(del,dir) =
 noBusesStep(del,dir,
  noBusesStep(del,dir,
   noBusesStep(del,dir,
    noBusesStep(del,dir,
     noBusesStep(del,dir,
      noBusesStep(del,dir,
       noBusesStep(del,dir,
        noBusesStep(del,dir,
         noBusesStep(del,dir,
          noBusesStep(del,dir,
           noBusesStep(del,dir,
            noBusesStep(del,dir,
             noBusesStep(del,dir,
              noBusesStep(del,dir,
               noBusesStep(del,dir,
                emptyStop(dir) & del & (![deadlock])))))))))))))))));
```

In order to get significant results, such property should be checked twice, with the *dir* parameter set to *airportBus* and *centreBus*, respectively. This can be done as follows:

```
Let noBusesFifteenSteps(del) =
   noBusesFifteenStepsDir(del,airportBus) |
   noBusesFifteenStepsDir(del,centreBus);
```

The formula can be instantiated with *del* set either the `noDelay` property defined above, in order to restrict checking to states (and paths) where no delay is introduced at any step by the chosen clumping mitigation strategy, or passing as *del* the `TT` logical constant (true at every point in every state) in order to avoid such restriction.

In the initial part of the generated model (first 40 minutes, just one branching point introduced, middle part of the model depicted in Figure 12), this property is true at some states for some stops, even with no delay added by the clumping mitigation strategy. Therefore, the problem was already present in the input

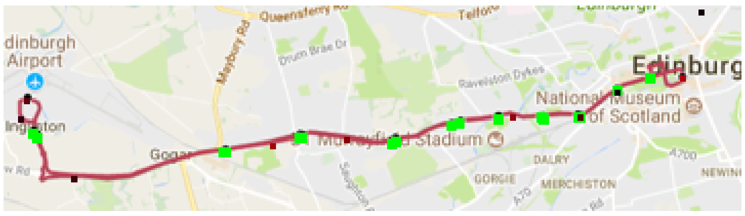


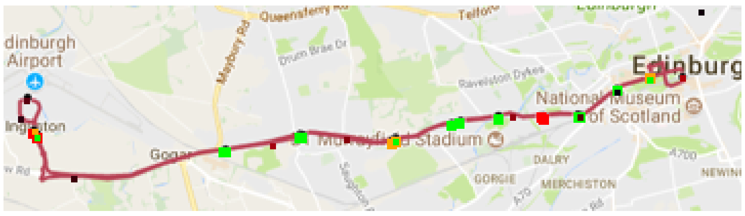
Fig. 16: Output at state 40 of the formulas `noBusesFifteenSteps(noDelay)` and `E F noBusesFifteenSteps(noDelay)`. No point is coloured by the model checker.

data. After 40 time steps, there is no stop satisfying this property without considering added delay, whereas there are several stops where the issue manifests, if added delay is taken into account. This indicates that the considered clumping mitigation strategy may indeed introduce large delays in the bus network. In Figure 16, we show the model checker output[8] in State 40 (highlighted in Figure 12), for the formulas `noBusesFifteenSteps(noDelay)` (in red) and `E F noBusesFifteenSteps(noDelay)` (in orange); indeed, no point is coloured in orange or red, as there is no future state exhibiting bus stops where no bus passes by for 15 minutes. Cumulative execution time for both formulas is about 5 minutes. The model checker output can be inspected by looking at the tree of system states, paired with the corresponding images. In doing so, starting from the initial state, one observes that evaluating the formulas at later time steps, some stops turn red, then orange, then red again, then green. In the initial state, some stations will not be reached by buses for quite some time, therefore they are red. Such issue could be considered spurious; however, later in time, as we already mentioned, there is a point in the input trace where the considered bus stops actually are not served by any bus for a long time. This is correctly highlighted by the model checker (the orange-coloured stops) in all states preceding the problem. Then the issue manifests, and stations are correctly coloured in red until a state is reached where the problem is no longer present.

In Figure 17, we show the model checker output at state 40, for the formulas `noBusesFifteenSteps(TT)` (in red) and `E F noBusesFifteenSteps(TT)` (in orange). Execution time is about 4 minutes. In this case, it is apparent that there are stops, especially in busy areas such as the airport and the city centre, that become problematic if the clumping mitigation strategy we are considering is used. Note that such results are obtained by looking at *all* system paths, therefore one should not think that all the potentially problematic stops become so in *the same* run of the system. Further inspection of the tree of system states can also be used to identify the choices that may lead to states with more stops exhibiting large delays.

Finally, since our strategy is designed to mitigate clumping, one may wonder if that really happens. We can check this by a variation of the method we used in formula `noBusesFifteenSteps(d)`. Intuitively, both clumping and “too few buses”

[8] States are numbered in the implementation and output for each state can be directly viewed without navigating the Kripke structure.



Map data copyright 2017 Google

Fig. 17: Output of the formulas `noBusesFifteenSteps(TT)` (in red) and `E F noBusesFifteenSteps(TT)` (in orange) at state 40.

are indications about the frequency of buses at stops. This explains the similarity between the two formulas.

```
Let clumping(del,dir) =
   del & busAtStop(dir) &
     (E X ((del & emptyStop(dir)) &
        (E X ((del & busAtStop(dir)) |
            (E X (del & busAtStop(dir)))))));

Let notDelayedClumping =
       touch(busStop,
          E F (clumping(noDelay,airportBus)
            | clumping(noDelay,centreBus)));

Let delayedClumping =
       touch(busStop,
          E F (clumping(delay,airportBus)
            | clumping(delay,centreBus)));
```

Formula `clumping(del,dir)`, similarly to `noBusesFifteenSteps`, only analyses states (and paths) that satisfy the property passed as parameter *del*. The additional parameter *dir* is necessary as, in the particular case of this route, buses going in opposite directions pass very close to stops of both directions; therefore one needs to check separately for each direction whether a clumping happens.

Once this is clarified, it is easy to see that formula `clumping(del,dir)` checks whether a bus of the specified direction is calling at a certain bus stop, then at step 2 no bus is calling at that stop, and finally, there is another bus calling at that stop in one or two subsequent steps. Such a situation is clearly an instance of the clumping problem; requiring the bus stop to be empty (i.e. no bus calling at it) for one step makes sure that the formula can be model checked in constant time with respect to the number of buses employed on the route: one does not need to check that the bus calling at the stop in step 2 is different from the one being there at step 1. Note that from step 3 on, any bus at the stop can be expected to be different from the one calling at the same stop in step 1, since the same bus is guaranteed to have left the stop in step 2. This way one does not need to instantiate the clumping formula for each possible "current" bus.

The size of the (simpler) `timeConglomerate` formula presented in Section 6 is linear in the number of buses, which results in too high model-checking time when

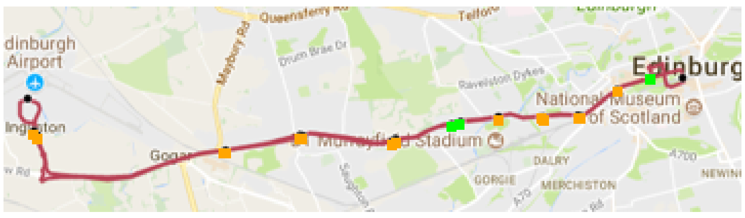


Fig. 18: Output of the formula `notDelayedClumping` (in orange).

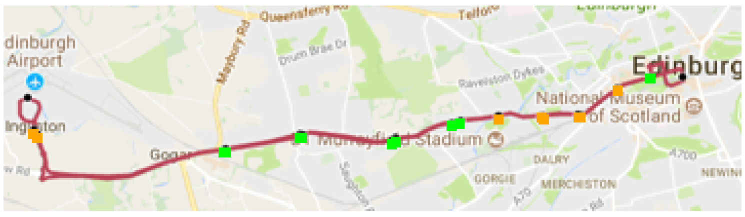


Fig. 19: Output of the formula `delayedClumping` (in orange).

checking for clumping that lasts more than 1 time step. Indeed the two formulas can be combined (detect if in the second step a different bus is present, then use the formula defined in this section to check clumpings with a length of more than one step), but in this particular experiment, possibly due to larger than usual distances between stops, we found very few clumpings of the type detected by the formula of Section 6. The two formulas `notDelayedClumping` and `delayedClumping` identify bus stops that will eventually contain clumpings, restricting analysis to states and paths without added delay (that is, the original trace) and to states and paths that have delay (that is, some buses have been waiting due to the waiting policy). The `touch` construct is used as a visual aid, to colour the whole bus stop, as the result of verification may colour a stop only partially in certain circumstances, due to a bus being held in a station and overlapping with it in the image. The two formulas take about 3 minutes each to be checked. Results (in the initial state) for the "no delay" case are reported in Figure 18, whereas results in the same state for the "delayed" case are reported in Figure 19, and clearly show effectiveness of the mitigation strategy. Almost all the bus stops exhibit clumping if no wait instruction is sent to buses.

8 Further Related Work

Spatial analysis has become an increasingly popular and varied area of research. Following is a selection of such research mainly addressing spatial issues in transportation illustrating the issues. A spatial time-series model for tracking planned journeys is presented in [43] although their main area of concern is vehicle speed forecasting rather than detection of outliers. The detection of outliers has been addressed by applying stochastic approaches. In [32] a method is presented for snapping GPS data onto a road network using a Hidden Markov Model. Noisy GPS data is identified as being the largest problem with snapping GPS readings onto road maps. The authors report that GPS signals can be reasonably modelled as a zero-mean Gaussian with a standard deviation of 10 metres. In [24] the authors present an approach to inferring the lane structure of roads from GPS data by fitting a mixture of Gaussians to GPS traces. This probabilistic approach naturally models the inherent noise in GPS data. In [35], instead, the authors apply the concept of functional depth to the identification of outliers in GPS observations. Outliers are identified by detecting curves rather than central values as in traditional statistical tests for comparing distributions.

The problems of headway computation are also considered in [36], where Monte Carlo simulation and time series analysis are used to evaluate a family of interpretations of an ambiguous regulation governing headway for frequent bus services.

From a more theoretical point of view, different forms of spatial logic have been proposed in computer science to refer to logics expressing properties of structured objects such as processes or data structures, in particular in the context of π-calculus (e.g. [10]) and mobile ambients with the related ambient logic (e.g. [12]). For example a binary logic operator has been introduced, $\Phi|\Psi$, that holds for a process P when this process is a parallel composition of two processes Q, satisfying Φ, and R, satisfying Ψ. Works such as [30,11] introduce notions of physical space in the context of process calculi; it is an interesting future work to connect this research line to spatio-temporal model checking. Furthermore, in a stochastic setting, the Mobile Stochastic Logic (MoSL) [21] has been proposed to predicate on mobile processes in models specified in StoKLAIM, a stochastic extension of KLAIM based on the tuple-space model of computations. Also logics for reasoning on *signals* have been enhanced with spatial aspects [27,8,25,33,4].

Other variants of spatial logics concern the symbolic representation of the contents of images, and, combined with temporal logics, for sequences of images [7]. The latter is based on a discretisation of the space of the images in rectangular regions and the orthogonal projection of objects and regions onto Cartesian coordinate axes such that their possible intersections can be analysed from different perspectives, whereas in [27] a linear spatial superposition logic is defined for the specification of emergent behaviour. The logic is applied in the context of medical image analysis for the recognition of patterns.

The spatial logic SLCS [14,15] and its spatio-temporal variant STLCS [13] used in the current paper, instead, addresses properties of discrete, graph-based models; such kinds of models include geographical maps, but also multi-dimensional images (see e.g. [5] for example applications in the medical imaging domain), and richer data structures such as *bigraphs* (see [39] for applications of further spatio-temporal extensions of SLCS, in the field of verification of *Cyber-Physical Systems*).

In the context of self-organising systems a spatial language has been proposed in [2] which can be used to assess spatial properties of system components to check desired global properties of the system against emergent global behaviours arising from local interactions among components. Spatial awareness is also an important issue in the context of pervasive ecosystems [22]. In that context the development of computational models is considered in which the entire structure of a pervasive system is modelled and constructed using an explicit spatial model, supporting multi-level spatial reasoning, and adapting autonomously to spatial interactions.

9 Conclusions

The use of a spatial model checker provides us with a sophisticated tool for checking complex properties over systems where location plays an important role, as it does in many collective adaptive systems (see for instance [40]). Using this tool we have been able to detect and correct a wide range of location-related errors in vehicle location data. By enhancing the logic and the model checker with a temporal perspective, the interplay of space and time has allowed us to define complex spatio-temporal formulas, predicating over the relation between points of a coloured image that evolves over traces or branching models. This methodology achieves a declarative approach to verification of spatio-temporal properties. This has a clear advantage over ad-hoc implementations of spatial analysis algorithms, because slight changes in the interpretation of given requirements do not impact the implementation of the verification tools, as only the logical formalisation of requirements needs to be changed.

From the experiment that we presented in Section 7, one can conclude that model sizes and model-checking times are reasonable for analysing simulations of bus holding strategies on real-world data using spatio-temporal model checking. Furthermore, the output format of the model checker makes it possible to explore the results of model checking interactively, by looking at the tree of all system states in the generated model, and looking at coloured maps for each state; a user of the system can therefore inspect the temporal evolution of the Boolean properties that are computed by the model checker. This can help users to detect correlations between the results of different spatio-temporal formulas. Such intrinsically "intelligent" process could as well be the object of machine learning procedures.

We note that our set-up does not yet cater for statistical analysis of the strategy taken into account. Indeed, our experiment could be repeated for several different time intervals of our input data, and the Boolean results could be turned into measures by counting the number of positive and negative results. This approach has already been demonstrated, in the context of bike sharing systems in [16], under the name of "statistical spatio-temporal model checking". Porting the methodology to the case study of bus networks will be the subject of future work. However, this is not straightforward, since the models we generate are branching, whereas traditional approaches to statistical model checking only operate on linear models. Future work will therefore need to address concerns such as estimating statistical confidence and accuracy of the results in statistical model checking of branching models.

Furthermore, it would be useful to get a more precise indication of when a wait instruction should be sent to a bus, e.g. by enumerating the traces that have no clumpings. An algorithmic way to obtain examples of such promising traces in which clumping is avoided is to generate a counterexample for `AF f`, where f is a formula indicating clumping. Counterexample generation for CTL (and more in particular for a subset of ACTL for which trace-like counterexamples can be generated in polynomial time) is a well-studied area (see for example [18, 9]). In particular, it is known that trace-like counterexamples can be generated in polynomial time using the algorithm described in [9]. The formula above is indeed of the type for which trace-like counterexamples exist. The generation of counterexamples in the context of spatio-temporal model checking is planned as future work; earlier experiments in this direction were made in the first prototype spatio-temporal model checker that was implemented before `topochecker`, but have not yet been ported to the new tool.

Recent work [15] is focused on defining *collective* variants of spatial and spatio-temporal properties; that is, the satisfaction value of a formula is defined on a set of points, rather than on a single point, so that the satisfaction value of a formula with respect to a set of points (a collective property) is not necessarily determined by the satisfaction values over the points composing the set (an individual property). Such interpretation of spatio-temporal logics is particularly motivated by the setting of collective adaptive systems and could as well bring interesting developments in the bus clumping case study.

An orthogonal, but nevertheless interesting, aspect of computation is the introduction of probability and of stochastic features. In future work, the *statistical spatio-temporal model checking* approach of [16] could be used to address questions like "how likely it is that a given strategy fixes the problem?" or "how frequently does the problem manifest itself, before and after applying a given strategy?". The task is however non-trivial, as one should first understand how statistical methods can be applied to the branching models that we derive from traces, whereas traditional statistical model checking only deals with *traces*, that do not contain non-deterministic choice points.

Acknowledgements The authors thank Bill Johnston of Lothian Buses and Stuart Lowrie of the City of Edinburgh council for providing access to the data related to bus positions.

References

1. Marco Aiello. *Spatial Reasoning: Theory and Practice.* PhD thesis, ILLC, University of Amsterdam, 2002.
2. Francesco Luca De Angelis and Giovanna Di Marzo Serugendo. Towards a spatial language for run-time assessments in self-organizing systems. In *2015 IEEE 9th International Conference on Self-Adaptive and Self-Organizing Systems, Cambridge, MA, USA, September 21-25, 2015*, pages 174–175. IEEE Computer Society, 2015.
3. Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.
4. Ezio Bartocci, Ebru Aydin Gol, Iman Haghighi, and Calin Belta. A formal methods approach to pattern recognition and synthesis in reaction diffusion networks. *IEEE Transactions on Control of Network Systems*, pages 1–1, 2016.
5. Gina Belmonte, Vincenzo Ciancia, Diego Latella, and Mieke Massink. From collective adaptive systems to human centric computation and back: Spatial model checking for medical imaging. In *Proceedings of the Workshop on FORmal methods for the quantitative Evaluation of Collective Adaptive SysTems, FORECAST@STAF 2016, Vienna, Austria, 8 July 2016.*, volume 217 of *EPTCS*, pages 81–92, 2016.
6. Mordechai Ben-Ari, Amir Pnueli, and Zohar Manna. The temporal logic of branching time. *Acta Informatica*, 20(3):207–226, 1983.
7. Alberto Del Bimbo, Enrico Vicario, and Daniele Zingoni. Symbolic description and visual querying of image sequences using spatio-temporal logic. *IEEE Trans. Knowl. Data Eng.*, 7(4):609–622, 1995.
8. Luca Bortolussi and Laura Nenzi. Specifying and monitoring properties of stochastic spatio-temporal systems in signal temporal logic. In *VALUETOOLS*, 2014.
9. Francesco Buccafurri, Thomas Eiter, Georg Gottlob, and Nicola Leone. On ACTL formulas having linear counterexamples. *J. Comput. Syst. Sci.*, 62(3):463–515, 2001.

10. Luis Caires. Behavioral and spatial observations in a logic for the π-calculus. In *Proceedings of the 7th International Conference on Foundations of Software Science and Computation Structures (FOSSACS'04)*, volume 2987 of *LNCS*, pages 72–87. Springer, 2004.
11. Luca Cardelli and Philippa Gardner. Processes in space. *Theoretical Computer Science*, 431(0):40 – 55, 2012. Modelling and Analysis of Biological Systems Based on papers presented at the Workshop on Membrane Computing and Bio-logically Inspired Process Calculi (MeCBIC) held in 2008 (Iasi), 2009 (Bologna) and 2010 (Jena).
12. Luca Cardelli and Andrew D. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *Proceedings of the 30th SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'00)*, pages 365–377, 2000.
13. Vincenzo Ciancia, Gianluca Grilletti, Diego Latella, Michele Loreti, and Mieke Massink. An experimental spatio-temporal model checker. In *Software Engineering and Formal Methods - SEFM 2015 Collocated Workshops*, volume 9509 of *Lecture Notes in Computer Science*, pages 297–311. Springer, 2015.
14. Vincenzo Ciancia, Diego Latella, Michele Loreti, and Mieke Massink. Specifying and Verifying Properties of Space. In Springer, editor, *The 8th IFIP International Conference on Theoretical Computer Science, TCS 2014, Track B*, volume 8705 of *Lecture Notes in Computer Science*, pages 222–235, 2014.
15. Vincenzo Ciancia, Diego Latella, Michele Loreti, and Mieke Massink. Model Checking Spatial Logics for Closure Spaces. *Logical Methods in Computer Science*, Volume 12, Issue 4, October 2016.
16. Vincenzo Ciancia, Diego Latella, Mieke Massink, Rytis Paskauskas, and Andrea Vandin. A tool-chain for statistical spatio-temporal model checking of bike sharing systems. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques - 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10-14, 2016, Proceedings, Part I*, volume 9952 of *Lecture Notes in Computer Science*, pages 657–673, 2016.
17. Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 53–71. Springer, 1986.
18. Edmund M. Clarke and Helmut Veith. Counterexamples revisited: Principles, algorithms, applications. In Nachum Dershowitz, editor, *Verification: Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday*, volume 2772 of *Lecture Notes in Computer Science*, pages 208–224. Springer, 2003.
19. Carlos F. Daganzo. A headway-based approach to eliminate bus bunching: Systematic analysis and comparisons. *Transportation Research Part B: Methodological*, 43(10):913 – 921, 2009.
20. Carlos F. Daganzo and Josh Pilachowski. Reducing bunching with bus-to-bus cooperation. *Transportation Research Part B: Methodological*, 45(1):267 – 277, 2011.
21. Rocco De Nicola, Joost-Pieter Katoen, Diego Latella, Michele Loreti, and Mieke Massink. Model checking mobile stochastic logic. *Theor. Comput. Sci.*, 382(1):42–70, 2007.
22. Simon A. Dobson, Mirko Viroli, Jose Luis Fernandez-Marquez, Franco Zambonelli, Graeme Stevenson, Giovanna Di Marzo Serugendo, Sara Montagna, Danilo Pianini, Juan Ye, Gabriella Castelli, and Alberto Rosi. Spatial awareness in pervasive ecosystems. *Knowledge Eng. Review*, 31(4):343–366, 2016.
23. E. Allen Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science (Vol. B)*, pages 995–1072. MIT Press, Cambridge, MA, USA, 1990.
24. Alireza Fathi and John Krumm. Detecting road intersections from GPS traces. In SaraIrina Fabrikant, Tumasch Reichenbacher, Marc van Kreveld, and Christoph Schlieder, editors, *Geographic Information Science*, volume 6292 of *Lecture Notes in Computer Science*, pages 56–69. Springer Berlin Heidelberg, 2010.
25. Ebru A. Gol, Ezio Bartocci, and Calin Belta. A formal methods approach to pattern synthesis in reaction diffusion systems. In *53rd IEEE Conference on Decision and Control*, pages 108–113, 2014.
26. Gianluca Grilletti. Spatio-temporal model checking: Explicit and abstraction-based methods. Master's thesis, Dipartimento di Matematica, Università di Pisa, 2016.
27. Radu Grosu, Scott A. Smolka, Flavio Corradini, Anita Wasilewska, Emilia Entcheva, and Ezio Bartocci. Learning and detecting emergent behavior in networks of cardiac myocytes. *Commun. ACM*, 52(3):97–105, 2009.

28. John J. Bartholdi III, Russell J. Clark, David W. Williamson, Donald D. Eisenstein, and Loren K. Platzman. Building a self-organizing urban bus route. In *Sixth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2012, Lyon, France, September 10-14, 2012*, pages 66–70. IEEE Computer Society, 2012.
29. John J. Bartholdi III and Donald D. Eisenstein. A self-coordinating bus route to resist bus bunching. *Transportation Research Part B: Methodological*, 46(4):481 – 491, 2012.
30. Mathias John, Roland Ewald, and Adelinde M. Uhrmacher. A spatial extension to the pi calculus. *Electronic Notes in Theoretical Computer Science*, 194(3):133 – 148, 2008. Proceedings of the First Workshop From Biology To Concurrency and back (FBTC 2007).
31. Roman Kontchakov, Agi Kurucz, Frank Wolter, and Michael Zakharyaschev. Spatial logic + temporal logic = ? In Marco Aiello, Ian Pratt-Hartmann, and Johan van Benthem, editors, *Handbook of Spatial Logics*, pages 497–564. Springer, 2007.
32. Julia Letchner, John Krumm, and Eric Horvitz. Trip Router with Individualized Preferences (TRIP): Incorporating personalization into route planning. In *Proceedings of the 18th Conference on Innovative Applications of Artificial Intelligence - Volume 2*, IAAI'06, pages 1795–1800. AAAI Press, 2006.
33. Laura Nenzi, Luca Bortolussi, Vincenzo Ciancia, Michele Loreti, and Mieke Massink. Qualitative and quantitative monitoring of spatio-temporal properties. In *Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22-25, 2015. Proceedings*, volume 9333 of *Lecture Notes in Computer Science*, pages 21–37. Springer, 2015.
34. Gordon F. Newell and Renfrey B. Potts. Maintaining a bus schedule. In *Proceedings of 2nd Australian Road Research Board*, volume 2, pages 388–393, 1964.
35. Celestino Ordoñez, Javier Martínez, Jose Rodríguez-Pérez, and Andria Reyes. Detection of outliers in GPS measurements by using functional-data analysis. *Journal of Surveying Engineering*, 137(4):150–155, 2011.
36. Daniël Reijsbergen and Stephen Gilmore. Formal punctuality analysis of frequent bus services using headway data. In András Horváth and Katinka Wolter, editors, *Computer Performance Engineering - 11th European Workshop, EPEW 2014, Florence, Italy, September 11-12, 2014. Proceedings*, volume 8721 of *Lecture Notes in Computer Science*, pages 164–178. Springer, 2014.
37. Minyan Ruan and Jie Lin. An investigation of bus headway regularity and service performance in Chicago bus transit system. In *Transport Chicago, Annual Conference, 14.*, 2009.
38. James G. Strathman, Thomas J. Kimpel, and Steve Callas. Headway deviation effects on bus passenger loads: Analysis of Tri-Met's archived AVL-APC data. Technical Report PR126, Portland State University, Centre for Urban Studies, Oregon, 2003.
39. Christos Tsigkanos, Timo Kehrer, and Carlo Ghezzi. Modeling and verification of evolving cyber-physical spaces. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, pages 38–48. ACM, 2017.
40. Christos Tsigkanos, Liliana Pasquale, Carlo Ghezzi, and Bashar Nuseibeh. Ariadne: Topology aware adaptive security for cyber-physical systems. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 729–732, May 2015.
41. Johan van Benthem and Guram Bezhanishvili. Modal logics of space. In Marco Aiello, Ian Pratt-Hartmann, and Johan van Benthem, editors, *Handbook of Spatial Logics*, pages 217–298. Springer, 2007.
42. Yiguang Xuan, Juan Argote, and Carlos F. Daganzo. Dynamic bus holding strategies for schedule reliability: Optimal linear control and performance analysis. *Transportation Research Part B: Methodological*, 45(1):1831 – 1845, 2011.
43. Zhixian Yan. Traj-ARIMA: A spatial-time series model for network-constrained trajectory. In *Proceedings of the Second International Workshop on Computational Transportation Science*, IWCTS '10, pages 11–16, New York, NY, USA, 2010. ACM.
44. Jiamin Zhao, Maged Dessouky, and Satish Bukkapatnam. Optimal slack time for schedule-based transit operations. *Transportation Science*, 40(4):529–539, 2006.