Neural Computing and Applications (2020) 32:405–426
https://doi.org/10.1007/s00521-018-3850-1

IWANN2017: LEARNING ALGORITHMS WITH REAL WORLD APPLICATIONS

An Intelligent Transportation System to control air pollution and road traffic in cities integrating CEP and Colored Petri Nets

Gregorio Díaz[1] · Hermenegilda Macià[1] · Valentín Valero[1] · Juan Boubeta-Puig[2] · Fernando Cuartero[1]

Received: 20 December 2017 / Accepted: 26 October 2018 / Published online: 12 November 2018

Abstract
Air pollution generated by road traffic in large cities is a great concern in today's society since pollution has an important impact on human health, even causing premature deaths. To address the problem, this paper presents an Intelligent Transportation System model based on Complex Event Processing technology and Colored Petri Nets (CPNs). It takes into consideration the levels of environmental pollution and road traffic, according to the air quality levels accepted by the international recommendations as well as the handbook emission factors for road transport methodology. This proposal, therefore, tackles a common problem in today's large cities, where traffic restrictions must be applied due to environmental pollution. CPNs are used in this work as a tool to make decisions about traffic regulations, so as to reduce pollution levels.

Keywords Intelligent control systems · Complex Event Processing · Event processing languages · Formal methods · Petri Nets

1 Introduction

The increase in vehicles in road traffic is a characteristic phenomenon of today's world, which means that related problems such as traffic accidents, pollution (air and noise), long travel times, etc., are increasing in the same way. Numerous reports have been published in order to determine the extent of the problem, and in particular this has led to the development of a new area of study, Intelligent Transportation Systems (ITS) [1]. ITS has emerged as an important element for both improving human life and the modern economy [2], with the main objective of optimizing road traffic by managing the capacity of the roads, improving driver safety, reducing energy consumption and improving the quality of the environment, among many others things. Moreover, an increase is expected in the development of ITS, integrating concepts such as big data, thus generating the new concept of Internet of Vehicles (IoV), as Xu et al. propose in [3], where a survey of applications of IoV and big data in autonomous vehicles is presented.

A key component for the study and development of ITS is traffic modeling, which provides a framework to better investigate and test the state of the road in real time and accurately predict future traffic. In general, a desirable model must meet the following requirements:

- It must be consistent with traffic flow.
- It must be flexible, using parameters that characterize the traffic flow, and be able to represent different situations and random changes in the traffic flow.
- It should be simple, but capable of capturing the information required in order to take decisions about traffic regulations.

✉ Gregorio Díaz
Gregorio.Diaz@uclm.es

Hermenegilda Macià
Hermenegilda.Macia@uclm.es

Valentín Valero
Valentin.Valero@uclm.es

Juan Boubeta-Puig
juan.boubeta@uca.es

Fernando Cuartero
Fernando.Cuartero@uclm.es

[1] School of Computer Science, University of Castilla-La Mancha, Campus Universitario s/n, 02071 Albacete, Spain

[2] Department of Computer Science and Engineering, University of Cadiz, Avda. de la Universidad de Cádiz 10, 11519 Puerto Real, Cádiz, Spain

In this context, we focus on traffic control in cities, taking into account the levels of environmental pollution according to the air quality levels accepted by the international recommendations [4]. Thus, we are tackling a common problem in large cities, where traffic restrictions must be applied due to pollution.

The methodology we use to design ITS is Complex Event Processing (CEP) [5] in combination with formal methods to model and test the proposed solutions [6]. CEP provides users with facilities for analyzing and correlating large volumes of data in the form of events with the aim of detecting relevant or critical situations for a particular domain in real time. To meet this objective, the conditions describing the situations of interest to be detected must be specified as event patterns. Patterns are implemented by using the languages provided by CEP engines, the so-called Event Processing Languages (EPLs), and once the patterns are defined, they can be deployed in the CEP engine in question [7].

Additionally, Petri Nets (PNs) [8] are a formalism which provides mathematical rigor and a graphical representation of the model, offering a better comprehension from a visual model and a mathematical underlying model in order to obtain important results about all its possible behaviors. Furthermore, Petri Nets are supported by tools, which allow us to simulate and analyze the behavior of a given system in a suitable manner.

The main aim of our proposal is to combine the use of CEP and Petri Nets to model and test ITS and, specifically, the city traffic flow, taking into account air pollution conditions. Thus, the contributions of this study are:

- A combination of CEP and Colored Petri Nets (CPNs) to provide an ITS.
- Definition of event patterns to detect high-risk situations produced by air pollutants.
- A model of traffic flow using CPNs.
- A methodology to test an ITS using the validation and verification features of CPNs.
- A realistic use case as a proof of concept, which also allows us to study the scalability of our proposal.

The structure of the paper is as follows. Section 2 presents the motivation of this work. Section 3 provides an overview of the CEP technology and the specific model of Petri Nets we use: CPNs. The city road model using CPNs is presented in Sect. 4 and the air quality and road traffic event patterns in Sect. 5. Section 6 presents the whole ITS system, integrating both the CPN model and the EPL patterns. This model is then applied to a real case study in Sect. 7, taking as reference the division into districts of Madrid, the capital city of Spain. Section 8 presents the related works, and finally, Sect. 9 presents our conclusions and lines of future work.

2 Motivation

In urban environments, road traffic can be a significant environmental problem due to the disproportionate exposure of citizens to environmental toxics, thus representing a public health problem. Air pollution remains one of the main factors related to preventable diseases and premature mortality in the EU. In 2010, it was estimated that air pollution in the EU caused more than 400,000 premature deaths. It was also the cause of preventable diseases, including respiratory conditions such as asthma, and exacerbated cardiovascular problems [9].

The greatest impact on human health occurs in urban areas, where air pollution levels are highest. Of particular concern is the health impact of exposure to atmospheric particulate matter of 2.5 micrometers ($PM_{2.5}$) and ozone (O_3). However, nitrogen dioxide (NO_2) and sulfur dioxide (SO_2) are also a concern, both on their own and as ozone precursors. Across Europe, it is estimated that 20–30% of the urban population is exposed to $PM_{2.5}$ with levels above the EU reference values, and 91–96% are exposed to more stringent levels than those of the World Health Organization [9]. At international level, the Organization for Economic Cooperation and Development (OECD) states that: "*Unless we clean the air, by the middle of the century one person will die prematurely every 5 s from outdoor air pollution*" [10].

According to [9], air pollutants can be classified as primary (emitted directly into the atmosphere) or secondary (formed in the atmosphere of precursor pollutants). The main primary air pollutants include primary PM, BC, sulfur oxides SO_x, NO_x (which includes NO and NO_2), NH_3, CO, methane (CH_4), benzopyrene (BaP) and hydrocarbons. Secondary air pollutants include secondary PM, O_3 and NO_2. The AQI index [11] reports daily air quality on the basis of five of these major pollutants.

Air pollutants may have a natural, anthropogenic or mixed origin, depending on their sources or the sources of their precursors. Emissions from motor vehicles contribute to air pollution in urban areas, and in many cities ensuring adequate air quality is a major problem. Road transport is the main source of air pollution in urban areas, and, therefore, there is a growing need to control current and future flow emissions as accurately as possible. As a result, a series of emissions models and emission factor databases have recently been developed. For instance, Yuan et al. [12] describe the complex way in which air pollution dispersion occurs in high density cities, such as Hong Kong.

Moreover, there is a relationship between the state of the traffic and the level of emission of pollutants. Borge et al. [13] presented a detailed study for the city of Madrid (Spain). This study was conducted by analyzing hourly

emissions from nearly 15,000 road segments distributed in 9 management areas covering Madrid City and surroundings. Traffic status was evaluated in four levels: free flow, heavy, saturated and stop and go. Significant quantitative information can be derived from this work, such as the relationship between the average speed vs the speed limit in order to establish the traffic level. Table 1 contains the traffic level ratios for a trunk road/primary city proposed in Borge et al.'s work.

3 Background

This section explains the background for both the CEP technology used for defining air quality and road traffic event patterns, and the CPN formalism.

3.1 Complex Event Processing

In CEP [14], a *situation* is an event occurrence or an event sequence that requires an immediate reaction. Events can be classified into two main categories: *simple events*, which are indivisible and happen at a point in time and *complex events*, which usually contain more semantic meaning and are obtained by processing a set of other events. Complex events can be derived from other events by applying or matching *event patterns*, i.e., templates where the conditions describing situations to be detected are specified. A *CEP engine* is the software used to match these patterns over continuous and heterogeneous event streams, and to raise real-time alerts after detecting them. These event patterns are implemented by using Event Processing Languages (EPLs). Further information about existing EPLs can be found in the survey by Cugola and Margara [15].

CEP is performed in 3 stages (see Fig. 1): (1) event capture—events are received and analyzed by CEP technology, (2) analysis—based on the event patterns previously defined in the CEP engine, the latter will process and correlate the information in the form of events in order to detect critical or relevant situations in real time, and (3) response—after detecting a particular situation, this will be notified to the system, software or device in question.

Table 1 Traffic level ratios in a trunk road/primary city

Level of traffic	Value	1-h average speed/speed limit
Free flow	1	> 0.87848
Heavy	2	0.75303–0.87848
Saturated	3	0.45306–0.75302
Stop and go	4	< 0.45306

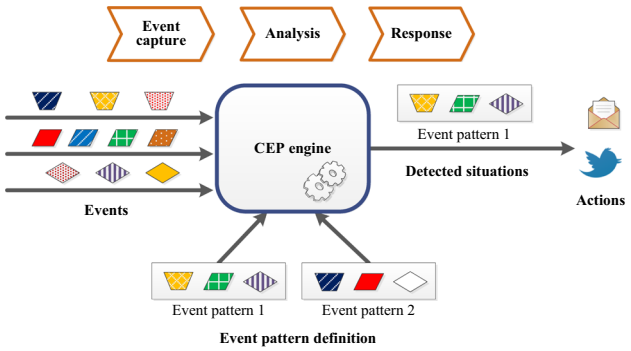
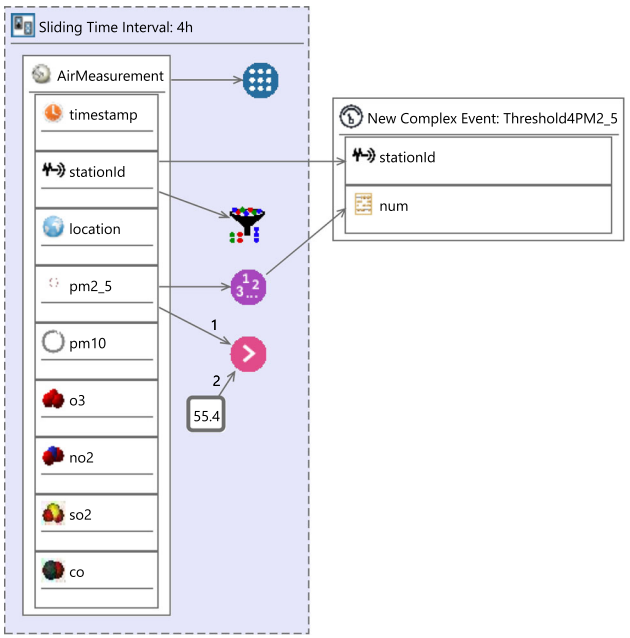


Fig. 1 Complex Event Processing stages

The main advantage of using this technology is that such important or critical situations can be identified and reported in real time, thus reducing latency in decision-making. It is worth noting that we have chosen Esper EPL [16] as EPL in this work, since this rich high-level processing language is more complete than others, providing more temporal and pattern operators for defining the situations of interest. For the sake of brevity, we refer to the particular language Esper EPL simply as EPL throughout the rest of the paper.

However, domain experts are usually unaware of CEP technology, and writing the event patterns code is a somewhat cumbersome task, which requires implementing the conditions to be met to detect relevant situations. Thus, we propose MEdit4CEP-CPN [6], a MEdit4CEP-based approach [7] extended by a Colored Petri Net (CPN) formalism, which supports the modeling, simulation, analysis and both syntactic and semantic validation of complex event-based systems. More specifically, MEdit4CEP-CPN provides domain experts with the ability to graphically model the event patterns (situations of interest) to be detected for a particular CEP domain. As an example, Fig. 2a shows the modeled event pattern in charge of detecting how many times the air quality 3-level has been exceeded (i.e., the *pm2_5* value is greater than 55.4 $\mu g/m^3$) per station in the last 4 h. Additionally, the editor validates the pattern syntax, automatically transforms the graphical pattern models into a CPN model, generates its corresponding CPN code executable by CPN tools [17], validates the pattern semantics and generates the Esper EPL code (see Fig. 2b) to be deployed in the final event-based system.

There are three important structural elements derived from EPL to consider in this work: the *schema*, which defines the event type structure; an *every* pattern operator, which provides us with all the events fulfilling a certain condition from the input data flow, and *sliding time data windows* for processing event information in time slides, as well as the *arithmetic* operators such as average, counter,



(a)

```
@Name("Threshold4PM2_5")
insert into Threshold4PM2_5
select a1.stationId as stationId,
   count(a1.pm2_5) as num
from pattern [every a1 =
   AirMeasurement(a1.pm2_5 > 55.4)].win:time(4 hours)
group by a1.stationId
```

(b)

Fig. 2 *Threshold4PM2_5* pattern. **a** Pattern model. **b** Pattern implementation in EPL

etc. Figure 3 shows three EPL extracts, where (a) specifies an event schema with two properties *propname1* and *propname2*, both of type *double*, (b) specifies an event pattern *Pattern1*, which *detects* the input events whose *propname1* is greater than 10.0 and (c) calculates the average of *propname2* values over the last 8 h.

3.2 Colored Petri Nets

A Petri Net (PN) is a bipartite directed graph, with two types of nodes, places (circles) and transitions (rectangles)

```
(a)  create schema event_name (propname1 double,
        propname2 double);

(b)  @Name('Pattern1')
     insert into Pattern1
     select a1.propname1
     from pattern [every a1 = event_name(a1.propname1>10.0)]

(c)  @Name('Pattern2')
     insert into Pattern2
     select avg(a1.propname2) as average
     from pattern [(every a1 = event_name)].win:time(8 hours)
```

Fig. 3 EPL basic schema and two patterns

[8]. Places and transitions can be connected by arcs, either place-transition (PT) or transition-place (TP) arcs (see Fig. 4). Let P be the set of places, T the set of transitions, $X = P \cup T$ (nodes) and $F \subseteq P \times T \cup T \times P$ the set of arcs. For any node $x \in X$ (place or transition), we define the preconditions and postconditions of x, denoted by ${}^\bullet x$ and $x^\bullet$, respectively, as follows: ${}^\bullet x = \{y \in X \mid (y,x) \in F\}$, $x^\bullet = \{y \in X \mid (x,y) \in F\}$.

Places usually represent states or system conditions, while transitions are the actions or events that produce changes in the system state. Arcs can have an associated weight (a natural number), by default 1. Places are then annotated by tokens to indicate system states. These tokens are usually depicted by dots or the number of tokens on the corresponding place. For example, a token on a place can indicate that the condition represented by this place is currently satisfied, or a number of tokens can indicate the number of processes waiting for a condition to occur, etc. The current state of the PN is thus defined by the set of tokens on every place, called the Petri Net marking, and a firing rule determines the conditions under which transitions are fired (executed) in order to change the current marking. Thus, for a transition to be fireable (*enabling condition*) all its precondition places must have at least as many tokens as the weight of the arc that connects them. The firing of a transition removes a number of tokens equal to the weight of the corresponding PT-arc from each precondition place and writes on its postcondition places as many tokens as indicated by the corresponding TP-arcs.

Colored Petri Nets (CPNs) are a well-known extension of Petri Nets. They extend the basic model with data information on the tokens. CPNs are supported by a widely used tool, CPN tools [17], which allows CPNs to be created, edited, simulated and analyzed. The notation described below is that used in this specific tool. In this paper, we only present an informal description of the CPN dynamical behavior. We omit the formal definitions, which can be found in [18, 19].

In CPNs, places have an associated *color set* (a data type), which specifies the set of token colors allowed at this place, so that tokens bring certain data information, according to the data type of its associated place. However, a place can have no attached information at all, as in the plain model. In this case, we indicate *UNIT* as the color set of the place. However, a place can now have as a color set, for instance, the set of integer numbers *INT*, a Cartesian product of two or more color sets as $INT2 = INT \times INT$, a string (*STRING*), etc. In this case, each token has an attached data value *(color)*, which belongs to the corresponding place color set. In CPN tools, the current number of tokens on every place is drawn in green on the right-hand side of the place circle, and the specific colors of these tokens are indicated by the notation n‘v, meaning that

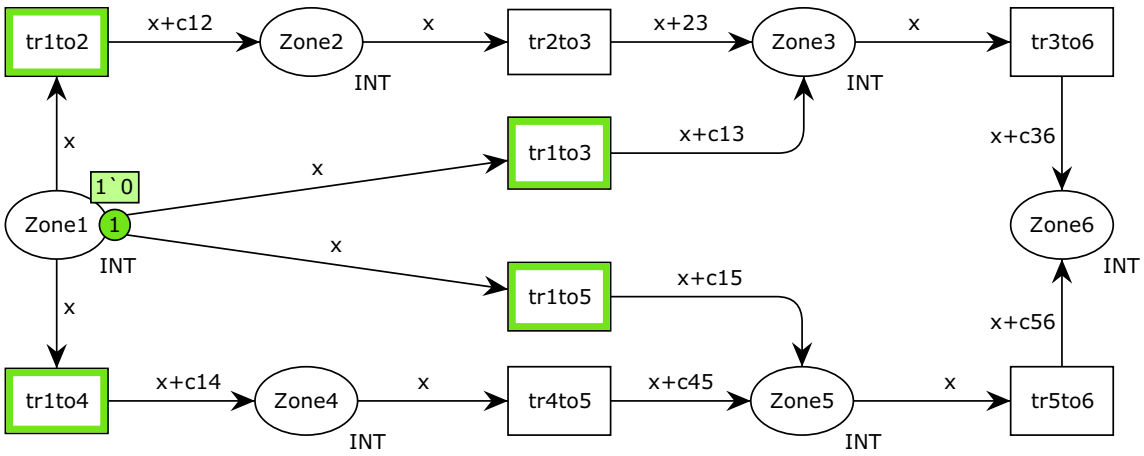


Fig. 4 A Marked Colored Petri Net

we have n instances of color v. Symbol "++" is used to represent the union of colors in CPN tools. Thus, a marking $1`3 + +3`2$ denotes that we have 1 token with value 3 and 3 tokens with value 2 on a place with color set *INT*.

The arc inscriptions are now extended to color set expressions, which are constructed using variables, constants, operators and functions. The arc expressions must evaluate to a color or multiset of colors in the *color set* of the attached place. Furthermore, transitions can have guards that can restrict their firing, which are Boolean expressions constructed by using the variables, constants, operators and functions of the model.

For any transition t with variables $x1, x2, \ldots$ on its input and output arc expressions, we call a *binding* of t an assignment of concrete values to each of these variables. A transition t is then *enabled* if there is a binding of t for which we have enough tokens on its precondition places matching the values of the corresponding inscriptions, and this binding makes true the guard of t.[1]

Thus, arc expressions are evaluated by assigning values to the variables, and these values are then used to select the tokens that must be removed or added when firing the corresponding transition. A transition t can then be fired when it is enabled for a binding b. When several transitions are enabled with their corresponding bindings,[2] the transition and binding selected for firing is chosen non-deterministically. The firing of t with a binding b removes the tokens on its precondition places matching with the values obtained for the corresponding arc expressions, and generates new tokens on its postcondition places with the values obtained for the associated arc inscriptions.

For instance, the CPN depicted in Fig. 4 models the different ways we have to travel from *Zone1* to *Zone6* in a city. Transitions *tritoj* represent transits from adjacent zones. Places *Zonei* have *INT* as color set, so variable x is integer. Arcs leaving these transitions *tritoj* are labeled with inscriptions $x + cij$, which means that the token produced will be delayed by *cij* time units (transit duration). Thus, all transitions *tr1toj*, for $j = 2, 3, 4, 5$ are enabled initially, so as to allow the movement from *Zone1* to *Zonej*. Let us assume *tr1to2* is fired. In this case, the token on *Zone1* is removed and place *Zone2* is marked with one token with value $c12$. Transition *tr2to3* can then be fired and *Zone3* is marked with one token with value $c12 + c23$. Finally, *tr3to6* can be fired, thus reaching *Zone6* with one token with value $c12 + c23 + c36$.

Following this same procedure for every CPN N with a given initial marking M_0 we can obtain all the markings reachable from it. We call $Reach(N, M_0)$ the set of all reachable markings from M_0 (*state space* of (N, M_0)). This set is of particular interest because it provides us with information about all the events that can occur in a system modeled by the considered net.

Finally, some CPN models can be very large, with a great number of both places and transitions, so the visualization of the whole model can be very difficult, not only because of the large number of places and transitions, but also due to the tangle of arcs crossing the net. The hierarchical features of CPN tools can then be used to split these models in several smaller pieces. These smaller pieces are called *pages* and can be linked by using substitution transitions and fusion sets. Substitution transitions refer to transitions that are replaced by subnets represented in other pages, while fusion sets are sets of places used in different pages, which are functionally identical and therefore correspond to the same place from a formal viewpoint. In this paper, we use fusion places to split the city map model in two pages, so the links between the pages are these common places, which have a blue fusion label on their left bottom corner.

[1] It is true by default, when no guard has been specified.

[2] Even the same transition with different bindings.

4 Modeling a city map with CPNs

Following the example depicted in Fig. 4, a city will be divided into zones, which are represented by places, labeled with the zone names: *Zonei*, $i = 1, 2, \ldots, n$, where n is the number of zones. Our goal is to obtain different routes to travel from *Zonea* to *Zoneb*. Transitions will represent movements from one zone to another by traversing some streets. Thus, transition *tritoj* captures the movement from *Zonei* to *Zonej*. These transitions have a Boolean guard, which will not allow the same zone to be traversed again to avoid cycles.

As an illustration, we show the main page of a simple city map CPN model in Fig. 5, which is an extension of the city map shown in Fig. 4, by allowing the reverse movements as well. The color set associated with places *Zonei* is defined as follows:

```
colset Zo = product INT * INTlist * INT;
```

which consists of a Cartesian product, where the first component corresponds to the target zone, the second component represents the route followed by the token to reach this place, as a list of traversed zones, from its starting zone to the current one, and the third component is the time spent using this route, expressed in minutes. As an example, see the marking of *Zone1* in Fig. 5: `(6,[1],0)`. This represents a car traveling from *Zone1* to *Zone6*, which is currently in *Zone1*, so no time has been spent yet. Variables x, y labeling the PT-arcs are of color set *Zo*, so they have the three components described above. TP-arcs are labeled with expressions that allow us to add the new step in the followed route and increase the time spent by the corresponding amount. Notice the transition guards `transit(x,i)`, which avoids our crossing the same zone twice and only allows a movement when a car has not reached its destination. Thus, `transit(x,i)` is only true when a car in Zone x can move to Zone i. This guard is defined as follows:

```
fun transit(x:Zo,i:INT) =
   noReturn(i,x) andalso not(isDest(x))
```

where function `noReturn` is used to avoid cycles and `isDest` checks whether the car has reached its destination. These functions are defined as follows:

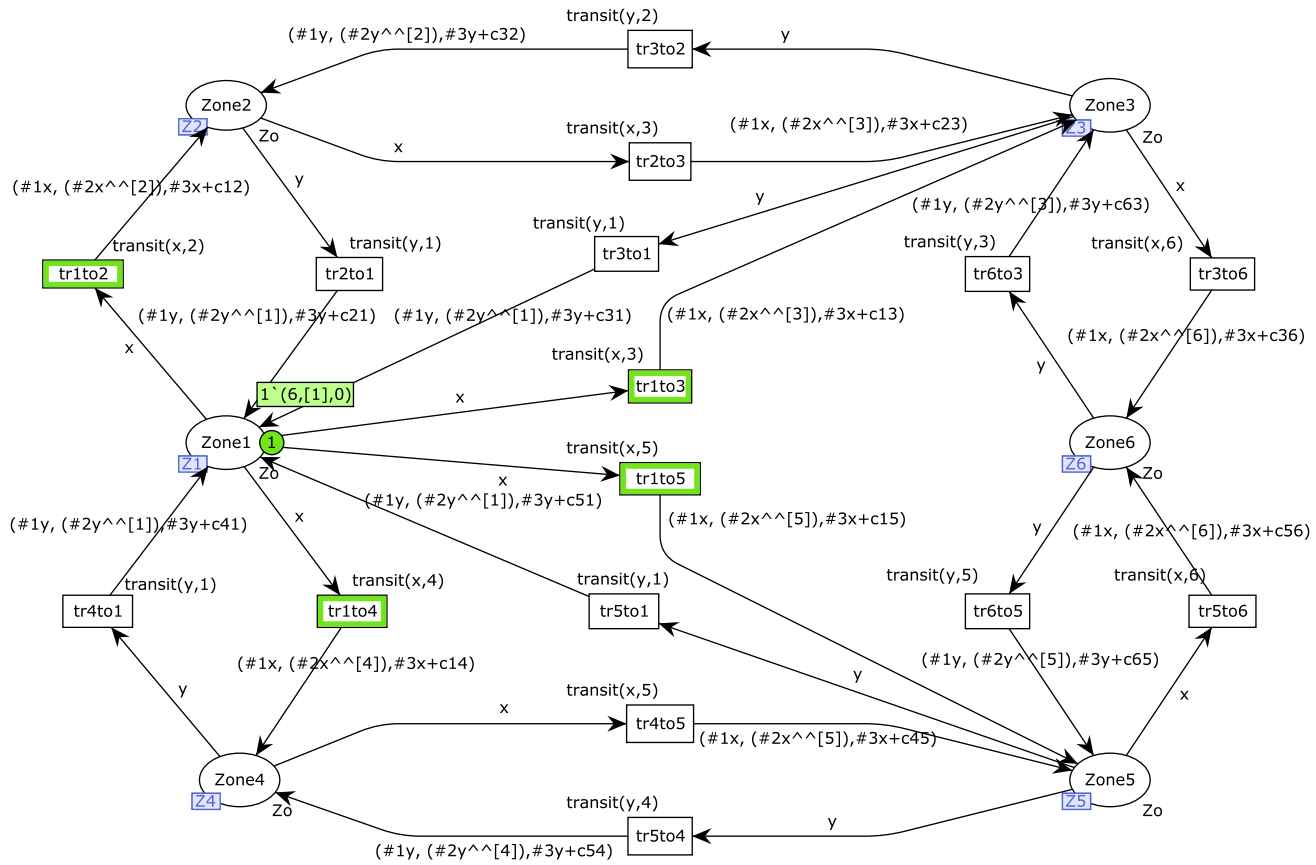


Fig. 5 Main city map CPN page

```
fun noReturn(i:INT,x:Zo)= not( mem (#2x) i)
fun isDest(x:Zo) = (hd(rev(#2x)) = #1x)
```

The operator "#i x" in both functions extracts the *ith* element of tuple x, that is, given the tuple (a,b,c), #1(a,b,c) evaluates to a. Function mem(l,i) is a Boolean function that checks whether element i is in the list l. Thus, noReturn checks whether *i* is in the second field of *x*, which contains the list of traversed zones. Function isDest reverses the list (function rev) and checks whether the head (function hd) is equal to the destination (first field of *x*).

For instance, let us consider the marking 1`(6,[1,2],3) in *Zone2*, which captures a car traveling from *Zone1* to *Zone6*, currently positioned in *Zone2* and having spent 3 min. To continue, two transitions could then be considered: *tr2to1* and *tr2to3*. However, transition *tr2to1* cannot be fired, since *Zone1* already belongs to the route list. Instead, transition *tr2to3* is enabled, since 3 does not belong to the list and the car has not reached its destination. Firing *tr2to3* removes the token (6,[1,2],3) from *Zone2* and generates a new token (6,[1,2,3],7) on *Zone3* (assuming $c23 = 4$). Notice the fusion tags on the left corner of places *Zonei*, which identify the shared places (fusion places) with the Destination CPN page.

Figure 6 depicts the Destination CPN page that completes the model, where the main element is place *Destination*. This is a sink place, where tokens come to finish the routes. It captures the information about the route followed and the total time spent on it. The color set of *Destination* is defined as follows:

```
colset De = product INTlist * INT;
```

which consists of a list (route to reach its destination) and an integer (total transit time).

4.1 State space analysis

The first technique we use for the system analysis is the state space graph, which provides us with all the possible system behaviors. Thus, we can get all the possible routes from a given initial location to a final location, which always allows the optimal solution to be found. Figure 7 shows the state graph obtained for the CPN depicted in Figures 5 and 6. The only place that initially has one token is *Zone1*, which represents a transit from *Zone1* to *Zone6*. There are 17 states in Figure 7. The notation on each state is the following: the state number at the top, and below it two numbers: the number of predecessors and the number of successors. We show the detailed state information in pink color for the terminal states[3] (those having 0 as number of successors). Only one place is marked with one token in all of these states, which corresponds to the location at which the route stops, either because the car reached its destination (place *Destination* marked) or because the car could not make any more movements (due to the restriction introduced of not allowing cycles). From the information on this token, we can obtain the route followed (list) and the total time spent on that route. For instance, *Zone4* is marked in state 8 with the route [1,5,4], which corresponds to one token that has traveled from Zone 1 to Zone 5, and then to Zone 4, which cannot go further, because there are no other outputs from *Zone4* than to return to either *Zone1* or *Zone5*. However, states 12, 13, 16 and 17 contain markings for which the *Destination* place is reached. Therefore, these states show us the routes that can be followed to cover the transit from *Zone1* to *Zone6*. We can now easily obtain the fastest route taking the state with the minimum time. Specifically, the fastest route in this case is highlighted at state 17, [1,2,3,6], which takes 9 min.

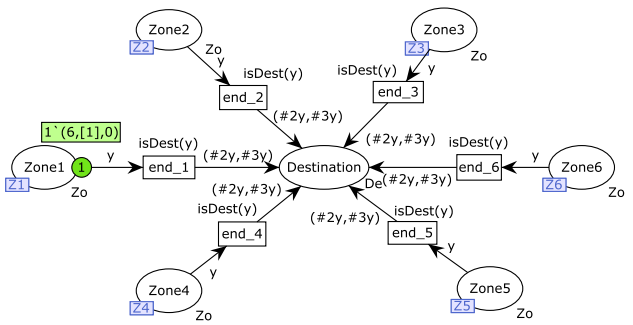


Fig. 6 Destination CPN page

4.2 Analysis via simulation

State space exploration is not always possible because the state graph may be huge in many cases, which makes generating it impossible due to time and resource restrictions. Simulation techniques can then be used in these cases to obtain a fast response, but as they do not usually cover all possible behaviors, the solution they provide is not always optimal. Simulations are based on experiment repetitions, so the solution returned is the best one obtained after a number of repetitions.

In our case, the experiment can be repeated by introducing not only one token at the starting zone, but a number of them, all with the same starting and destination zone, so as to cover as many paths as possible. Simulations are performed automatically, and we cannot control the CPN tools simulator engine, so there is no way to avoid path repetitions. As a consequence, the number of tokens at

[3] A state is said to be terminal when no movements can be made from it.

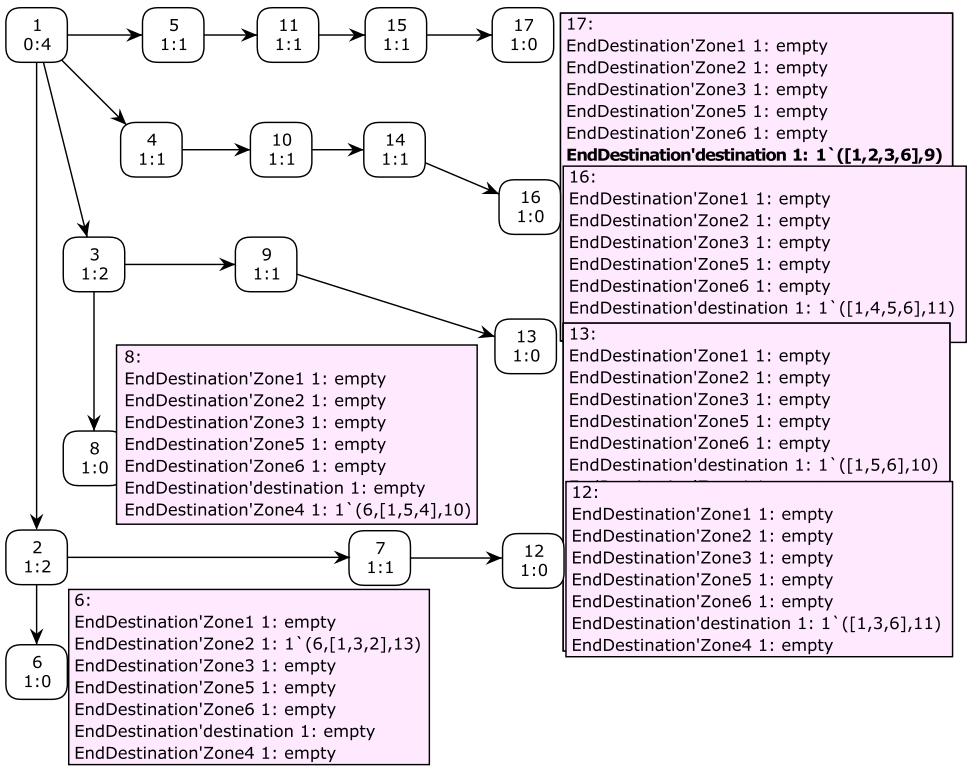


Fig. 7 State Space Analysis to obtain all available routes for a car traversing the city from Zone 1 to Zone 6

the initial zone is chosen as a model constant, independent of the map structure.

Each token can follow a different path, so place *Destination* will possibly become marked with many tokens, those that have reached the destination zone. From these tokens, we take the one providing us with the minimum time, which is then the best route among those followed by the tokens.

Figure 8 depicts this situation, where, from the marking of place *Destination*, we can conclude that 18 tokens have reached their destination in 9 min. This is the best route for this simulation, since there is no other simulated route taking less time. This is actually the same route returned by the state space exploration technique, and the other routes in place *Destination* also correspond to the successful final states obtained with the state graph. The correspondence between the graph states and simulation routes is presented in Table 2. Routes **a** to **d** are all successful, route **a** being the fastest solution. By contrast, routes **e** and **f** lead to a deadlock.

Simulation times are noticeably shorter than the times required to construct the state space, especially for large CPNs. Thus, it is a technique that quickly provides good, but possibly not optimal, solutions. In Sect. 7, we will combine both techniques in order to improve state space exploration by using the branch and stop options of CPN tools. These options allow us to prune the exploration when a route is more expensive than one that has already been computed or one known by simulation.

Finally, simulations can be expanded by including tokens on different zones, so as to check different transit routes simultaneously. Transit routes with stops can also be obtained with these techniques, by dividing the route into legs that are analyzed separately.

5 Event pattern modeling

This section explains the event patterns modeled and automatically implemented, by using our MEdit4CEP-CPN editor [6], to detect situations of interest in two domain applications: air quality and road traffic.

5.1 Air quality

For simplicity, in the following description we only consider one important pollutant: $PM_{2.5}$. As explained in Sect. 2, around 20–30% of the urban population in Europe is exposed to this pollutant with levels above the EU reference values, causing numerous premature deaths and preventable diseases. In any event, the following methodology would be similarly applied for the other pollutants.

The US Environmental Protection Agency (EPA) provides information on the ranges of each pollutant in a particular air quality level. Based on the EPA technical

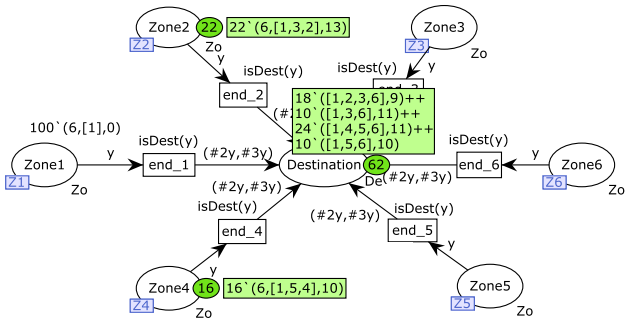


Fig. 8 Simulation result for 100 tokens from *Zone1* to *Zone6*

Table 2 Mapping graph states and simulation routes

Route	Place	Space state	Simulation
a	*Dest*	17: 1‘([1,2,3,6],9)	18‘([1,2,3,6],9)
b	*Dest*	16: 1‘([1,4,5,6],11)	24‘([1,4,5,6],11)
c	*Dest*	13: 1‘([1,5,6],10)	10‘([1,5,6],10)
d	*Dest*	12: 1‘([1,3,6],10)	10‘([1,3,6],11)
e	*Zone4*	8: 1‘(6,[1,5,4],10)	16‘(6,[1,5,4],10)
f	*Zone2*	6: 1‘(6,[1,3,2],13)	22‘(6,[1,3,2],13)

information, a classification is made calculating the average value of a pollutant across 1 h, 8 h or 24 h, depending on the type of pollutant. For instance, for $PM_{2.5}$, the average value over a 24-h period is required. Once we have this average value, we can report the air quality level by taking the range to which the value belongs (see Table 3).

The EPA also defines a global level for air quality, the *Air Quality Index* (AQI) [11], which is calculated as the highest of all the pollutant levels in a location at a specific time, so as to obtain one of six air quality levels: *Good*, *Moderate*, *Unhealthy for Sensitive Groups*, *Unhealthy*, *Very Unhealthy* and *Hazardous*.

According to this EPA technical information, the *AirMeasurement* domain and a set of event patterns have been graphically modeled by using MEdit4CEP-CPN, to detect the AQI level at a particular location. We assume data are received in this location according to such a domain, which has been modeled and transformed into EPL code as follows:

```
create schema AirMeasurement(
   timestamp string, stationId string, location string,
       pm2_5 float, pm10 float, o3 float, no2 float,
       so2 float, co float);
```

More specifically, this EPL schema defines the event information required for the air quality measurements. It contains the time at which the measurement is taken, the station and location identifiers and the pollutants included in the AQI index ($PM_{2.5}$, PM_{10}, O_3, NO_2, SO_2 and CO).

Once the *AirMeasurement* domain is designed, the event pattern editor is automatically reconfigured for this domain. Figure 9a shows the design of a pattern that computes the average value for $PM_{2.5}$ at every location based on the $PM_{2.5}$ measurements received during the last 24 h. Thus, from all the simple events of *AirMeasurement* for a same location the average value for $PM_{2.5}$ is obtained, and a new complex event with the *stationId* and the computed average *value* is created and inserted into the flow *PM2_5Avg*, so as to have all *PM2_5Avg* average values obtained over the time period. These average values are computed as they are received by using time *sliding* data windows.

Once the pattern is modeled and syntactically validated, the EPL code automatically generated for the *PM2_5Avg* pattern is shown in Fig. 9b.

In parallel, we monitor the *PM2_5Avg* events, to check the level of $PM_{2.5}$. For this purpose, we have defined 6 additional patterns to detect when $PM_{2.5}$ is Good, Moderate, Unhealthy for Sensitive Groups, Unhealthy, Very Unhealthy and Hazardous. For instance, Fig. 10a shows the modeled *PM2_5Moderate* pattern. It is detected when the average value of $PM_{2.5}$ is greater than or equal to 12.1, and smaller than 35.5. In this case, a new complex event with the station Id, the level name (*PM2_5Moderate*) and a level number (2 has been assigned for $PM_{2.5}$ moderate) is created and inserted into the *PollutantLevel* flow. The EPL code generated for this pattern is shown in Fig. 10b.

PM2_5Good, *PM2_5UnhealthyForSensitiveGroups*, *PM2_5Unhealthy*, *PM2_5VeryUnhealthy* and *PM2_5Hazardous* patterns are defined analogously according to the intervals for average $PM_{2.5}$ values described in Table 3. The corresponding complex events will be inserted into the *PollutantLevel* flow, with *levelNumber* 1, 3, 4, 5 and 6, respectively.

The *AirQualityLevel* pattern has then been modeled as indicated in Fig. 11a. This pattern selects the maximum air quality level detected during 5-min batching windows for a particular station and establishes this level as the air quality level for the station, inserting it in the *AirQualityLevel* flow.

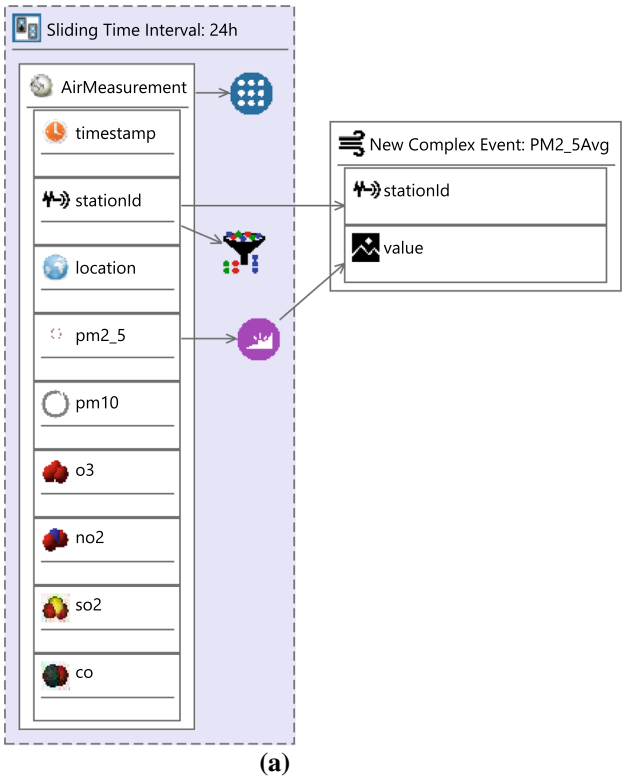
5.2 Road traffic

In order to automatically detect the current level of traffic in a particular city, we modeled a set of event patterns. These are defined according to the traffic ratios (see Table 1) based on the statistic speeds and speed limits reported by the Handbook Emission Factors for Road Transport (HBEFA) methodology.

By using the MEdit4CEP-CPN editor, as in the previous air quality scenario, the *Traffic* domain was modeled and transformed into EPL code as follows:

Table 3 AQI categories

Air quality category			Pollutants					
Name	L	Color	NO_2 (ppb) 1 h	SO_2 (ppb) 1 h	CO (ppm) 8 h	O_3 (ppm) 8 h	$PM_{2.5}$ ($\mu g/m^3$) 24 h	PM_{10} ($\mu g/m^3$) 24 h
Good	1	Green	0–53	0–35	0.0–4.4	0.000–0.054	0.0–12.0	0–54
Moderate	2	Yellow	54–100	36–75	4.5–9.4	0.055–0.070	12.1–35.4	55–154
Unhealthy for sensitive groups	3	Orange	101–360	76–185	9.5–12.4	0.071–0.085	35.5–55.4	155–254
Unhealthy	4	Red	361–649	186–304	12.5–15.4	0.086–0.105	55.5–150.4	255–354
Very unhealthy	5	Purple	650–1249	305–604	15.5–30.4	0.106–0.200	150.5–250.4	355–424
Hazardous	6	Maroon	1250–2049	605–1004	30.5–50.4	> 0.200	250.5–500.4	425–604



(a)

```
@Name("PM2_5Avg")
insert into PM2_5Avg
select a1.stationId as stationId,
   avg(a1.pm2_5) as value
from pattern [(every a1 =
   AirMeasurement)].
   win:time(24 hours)
group by a1.stationId
```

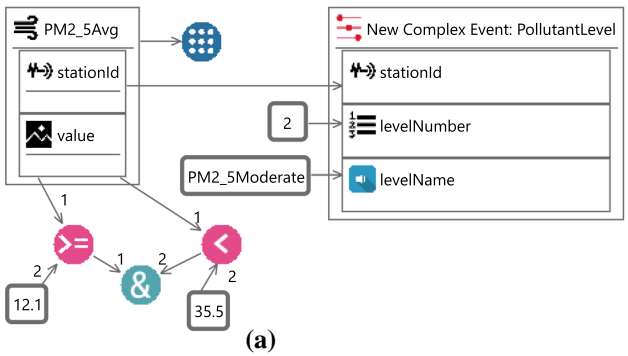
(b)

Fig. 9 *PM2_5Avg* pattern. **a** Pattern model. **b** Pattern implementation in EPL

```
create schema Traffic(
   timestamp string, stationId string, location string,
       speed float);
```

This EPL schema defines the event information required for the traffic measurements which are taken every 5 min by the Madrid City Council. More specifically, a *Traffic* event is generated every 5 min containing the average speed of all the vehicles that pass by a particular station during this time, the time stamp at which the event is created, the station identifier and the station location.

Once the *Traffic* domain is designed, the event pattern editor is automatically reconfigured for this domain, which allows us to define the corresponding patterns. As an illustration, Fig. 12a shows the design of a pattern that computes the last 1-h average speed at every station. Thus, from all the simple events of *Traffic* that have passed by a station, the average speed value is obtained, and a new



(a)

```
@Name("PM2_5Moderate")
insert into PollutantLevel
select a1.stationId as stationId,
   2 as levelNumber,
   'PM2_5Moderate' as levelName
from pattern [(every a1 =
   PM2_5Avg(a1.value >= 12.1 and
   a1.value < 35.5))]
```

(b)

Fig. 10 *PM2_5Moderate* pattern. **a** Pattern model. **b** Pattern implementation in EPL

complex event with the *stationId* and the computed average *value* is created and inserted into the *SpeedAvg* flow.

Once the pattern is modeled and syntactically validated, the EPL code automatically generated for the *SpeedAvg* pattern is shown in Fig. 12b.

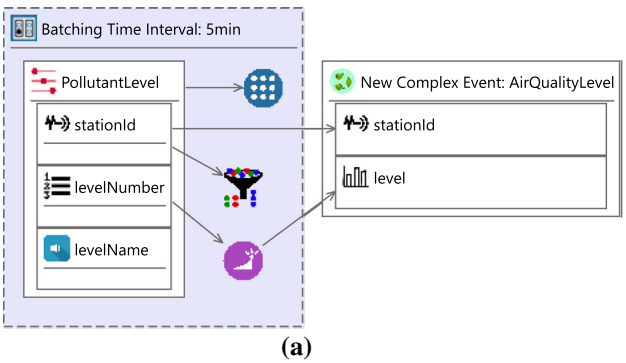
Moreover, we monitor the *SpeedAvg* events in order to check the traffic level, according to Table 1. For this purpose, we have defined 4 additional patterns to detect when the traffic level is free flow, heavy, saturated and stop and go. As an example, Fig. 13a depicts the modeled *SaturatedTraffic* pattern. It is detected when the average speed value divided by 50 Km/h (the speed limit in a city) is greater than or equal to 0.45306, and smaller than or equal to 0.75302. In this case, a new complex event with the station Id and the level number (3 has been assigned for the saturated traffic level) is created and inserted into the *SaturatedTraffic* flow. The EPL code generated for this pattern is shown in Fig. 13b.

6 ITS for traffic control by using CEP and CPNs

CEP technology provides us with air pollution and traffic condition levels, which can then be used in the city map CPN model presented in Sect. 4 to make decisions about traffic regulations. In this scenario, we use synthetic data to simulate this information flow, which is assumed to be provided by a CEP engine. The integration of both technologies, thus allowing the CPN to provide traffic regulations conclusions in real time, will be a matter of future research, as indicated in the conclusions section, and also following some ideas about the learning of the proposed system that were presented by Li et al. in [20].

We start from the city map CPN model presented in Fig. 5, which is now enriched in order to manage the two situations of interest indicated in the previous section. The intention is to make decisions about traffic control by closing and opening the connections between zones according to the levels of traffic and pollution. Closing an area may seem an extreme decision, especially if it is an access to the city. However, it is a measure that can be adopted in large cities, as occurs in the case of Madrid. According to Article 35 of the Sustainable Mobility Ordinance, approved by the Madrid City Council, extraordinary measures to restrict traffic and parking vehicles on urban roads may be enacted either during episodes of high air pollution or for reasons of road safety and severe traffic congestion [21]. Taking this ordinance as reference, connections will be closed when one of the following conditions occurs:

1. The pollution level is higher than 3, i.e., Unhealthy, Very Unhealthy or Hazardous.
2. The pollution level is Unhealthy for Sensitive Groups (3), but there have been 16 peaks of $PM_{2.5}$ with values higher than 55.4 $\mu g/m^3$ in the last 24 h.
3. The Pollution level is Unhealthy for Sensitive Groups (3) and traffic level is either *Saturated* or *Stop* and *go* (levels 3 and 4, respectively).

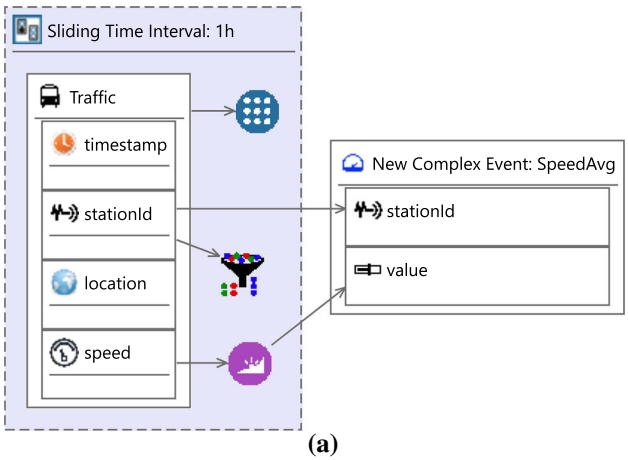


(a)

```
@Name("AirQualityLevel")
insert into AirQualityLevel
select a1.stationId as stationId,
   max(a1.levelNumber) as level
from pattern [(every a1 =
   PollutantLevel)].
   win:time_batch(5 minutes)
group by a1.stationId
```

(b)

Fig. 11 *AirQualityLevel* pattern. **a** Pattern model. **b** Pattern implementation in EPL

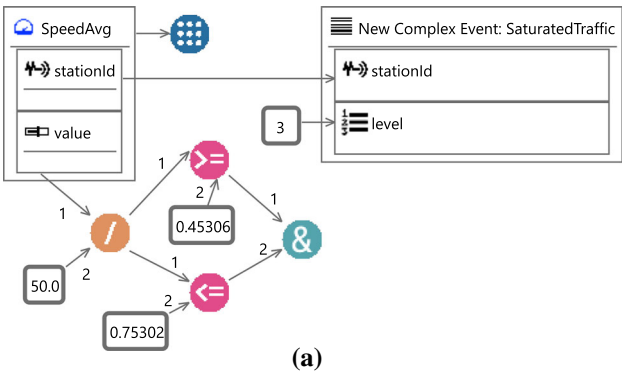


(a)

```
@Name("SpeedAvg")
insert into SpeedAvg
select a1.stationId as stationId,
   avg(a1.speed) as value
from pattern [(every a1 = Traffic)].
   win:time(1 hours)
group by a1.stationId
```

(b)

Fig. 12 *SpeedAvg* pattern. **a** Pattern model. **b** Pattern implementation in EPL



(a)

```
@Name("SaturatedTraffic")
insert into SaturatedTraffic
select a1.stationId as stationId,
   3 as level
from pattern [(every a1 = SpeedAvg(
   ((a1.value / 50.0) >= 0.45306 and
   (a1.value / 50.0) <= 0.75302)))]
```

(b)

Fig. 13 *SaturatedTraffic* pattern. **a** Pattern model. **b** Pattern implementation in EPL

These specific conditions are encoded in our CPN model by the Boolean function `close`:

```
fun close(a:int,p:int,l:int) =
  if a>3 orelse (a=3 andalso p>15) orelse
  (a=3 andalso l>2) then true else false
```

Parameter *a* represents the Air Quality level, obtained by the `AirQualityLevel` pattern (see Fig. 11), *p* stands for the number of peaks, obtained by the `Threshold4PM2_5` pattern (see Fig. 2) and *l* is the traffic level, obtained by patterns `FreeflowTraffic`, `HeavyTraffic`, `SatturatedTraffic` and `StopAndGoTraffic` (see Fig. 13 for *SaturatedTraffic level* pattern).

Furthermore, traffic levels are computed taking into account the average speeds of cars in relation to the limits, so we can now use the current traffic level to compute the average times for the transits between adjacent zones, using the following function:

```
fun traffic_time(l:int,t:int) =
   case l of
         1 => round(real t/1.0)
       | 2 => round(real t/0.8)
       | 3 => round(real t/0.5)
       | 4 => round(real t/0.1)
```

Parameter `l` is the traffic level and `t` the average time for a transit between two adjacent zones. This function takes into account the ratios between average and limit speeds provided in Table 1, establishing how the transit time is increased for levels 2, 3 and 4. These times are actually increased in an inverse proportion of 80%, 50% and 10%, respectively. These percentages capture the average proportion reduction in traffic speed according to Table 1.

A function `genLevels` has been implemented, using four auxiliary functions, to randomly produce level and peak values in the CPN.

```
fun genLevels()=1`(genAQLlevel(),genPeaks(),
                   genTraffic(),genTraffic())
```

Function `genAQLlevel` generates Air Quality Levels, `genPeaks` generates the peaks and `genTraffic` the traffic levels for transits between *Zonei* to *Zonej* and vice versa. Function `genAQLlevel` is based on a custom distribution, where levels 2 and 3 have the highest likelihood (40% and 35%), levels 1 and 4 are almost the same (9% and 10%), level 5 is rare (5%), and level 6 is very rare (1%). It is then defined as follows:

```
fun genAQLlevel() =
let
   val b = discrete(1,100)
in
   if (b <= 9) then 1
   else if (b >  9 andalso b <= 49) then 2
   else if (b > 49 andalso b <= 84) then 3
   else if (b > 84 andalso b <= 94) then 4
   else if (b > 94 andalso b <= 99) then 5
   else 6
end;
```

where function `discrete(1,100)` generates a random value between 1 and 100, which is then used to produce the final result. In a similar way, function `genPeaks` produces the peaks, which are evenly distributed between 1 and 20 occurrences, so it is defined as follows:

```
fun genPeaks() = discrete(1,20)
```

The generation of traffic levels is also based on a custom distribution, where levels 1 and 2 occur often (40% and 50%), 3 is rare (8%), and 4 hardly occurs (2%):

```
fun genTraffic() =
let
   val b = discrete(1,100)
in
   if (b <= 40) then 1
   else if (b >40 andalso b <=90) then 2
   else if (b >90 andalso b <= 98) then 3
   else 4
end;
```

Function `genLevels` is used in the new city map CPN model (see Fig. 14) to feed places *Levelij* by the firing of a new transition *init_g*. Place *pinit* is initially marked with one single token, so *init_g* only fires once. Its firing generates one token on each place *Levelij*, with the 4 values previously mentioned, namely Air Quality level, Peaks and Traffic level in both directions.

Thus, the `Levels` color set is defined as follows:

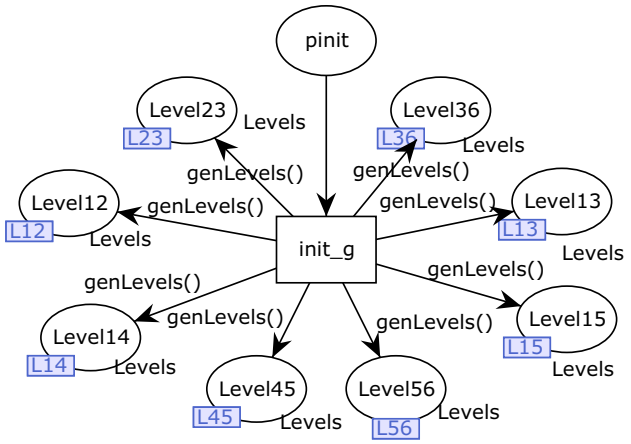


Fig. 14 Producing initial level information

```
colset Levels = product INT*INT*INT*INT;
```

For instance, we could obtain one token `1`(3,15,2,3)` as initial marking for a place *Levelij*, which corresponds to an AQI 3 level, 15 peaks and traffic levels 2 (heavy) for the transit *Zonei* to *Zonej* and 3 (Saturated) for the opposite direction.

The city map CPN model is then modified as indicated in Fig. 15. Transit transitions (*tr2to3* and its reverse *tr3to2* in the CPN piece shown in the figure) now use the level information provided by *init_g*, so guards `isOpen(3,x,a,p,l1)` and `isOpen(2,y,a,p,l2)` are used to check whether the transit is open or closed, taking into account pollutants and traffic conditions.

These guards are defined as follows:

```
fun isOpen(i:int,y:Zo,a:int,p:int,l:int) =
       noReturn(i,y) andalso not(close(a,p,l))
       andalso not(isDest(y))
```

As an illustration, consider the markings shown in Fig. 15 for places *Zone2*, *Zone3* and *Level23*. Transition *tr3to2* is closed, since the Boolean expression `(a=3 andalso l2> 2)` is true and therefore `close` returns true. However, transition *tr2to3* is open, since none of the close conditions is satisfied.

Notice also the use of function `traverseSection` in the arcs from transit transitions to their destination zones (see Fig. 15). This function is implemented by using the *traffic_time* function commented above, as follows:

```
fun traverseSection(i:INT,x:Zo, t:INT, l:INT)=
       (#1x, (#2x^^[i]),#3x+traffic_time(l, t))
```

which adds the current leg[4] to the path followed and increases the total time by the amount computed by function *traffic_time*. For instance, taking again the marking in Fig. 15, we have obtained $traverseSection(3, x, c23, l1) = (6, [1, 2, 3], 8)$.

Figure 16 indicates the changes for the connections with the *Destination* place. In this figure, we have considered 100 tokens in the *Zone1* place traveling from *Zone1* to *Zone6*. This choice of 100 tokens for the simulations has turned out to be an appropriate value to obtain the minimal route, i.e., some tokens have been able to reach the *Zone6* place. Notice the loop arcs between the *Zonei* places and the *end_i* transitions, which are now introduced so as to obtain the minimal route in the *Destination* place, instead of all the possible routes from *Zone1* to *Zone6*. Thus, only one initial token is now required on *Destination* for the route under study, which initially has a very high value (*tmax*) as total transit time (second field). As new tokens arrive at *Destination*, this value is updated with the minimum value between the old one and that of the token that has just arrived. For this purpose, function `mini` is defined, which compares the time stored on the token on *Destination* with the time obtained from a route, taking the minimum one:

```
fun mini(y:Zo,z:De)=
   if (#3y < #2z) then 1`(#2y,#3y) else 1`z
```

Function `isDestiny` extends function `isDest` introduced in Sect. 4. This function enables transition *end_i* when the token represented by parameter `y` has reached its destination (`isDest`), as previously mentioned. Furthermore, `isDestiny` checks whether the initial and target zones are the same in `y` and `z`.

```
fun isDestiny(y:Zo,z:De)=
       isDest(y) andalso
       hd(rev(#2y))=hd(rev(#1z)) andalso
       hd (#2y) = hd (#1z)
```

Should we need to compute a different route, we would only change the initial marking on *Destination* to the corresponding one `1`([i,j],tmax)`, writing the following initial marking on place *Zonei*: `100`(j,[i],0)`. Moreover, we can compute several routes at the same time by marking the *Zonei* places and the *Destination* place with

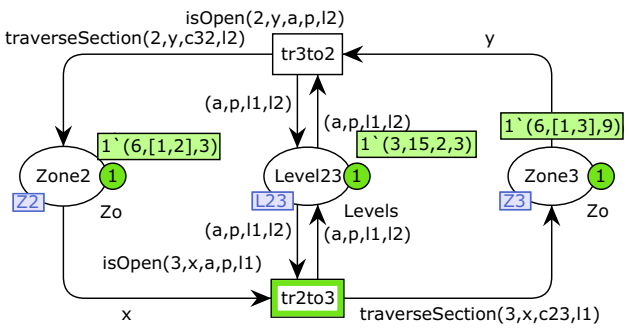


Fig. 15 Transits between *Zone2* and *Zone3*

[4] `^^[i]` adds the current zone `i` to the list.

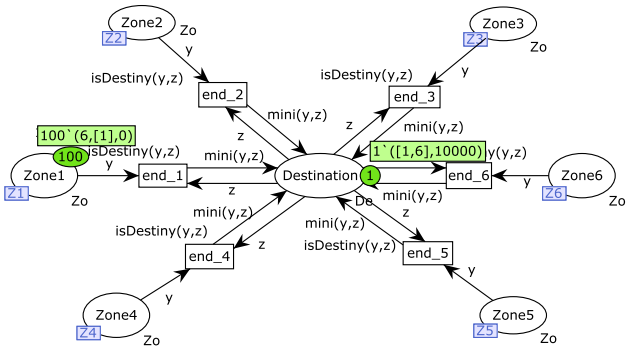


Fig. 16 New Destination CPN page

the corresponding tokens. For instance, we can compute a route from *Zone1* to *Zone3* and a route from *Zone3* to *Zone6* writing the marking `1`([1,3],tmax)++ 1`([3,6],tmax)` at *Destination*, token `100`(3,[1],0)` at *Zone1* and token `100`(6,[3],0)` at *Zone3*.

Figures 17 and 18 show the pollution and traffic levels obtained in a simulation and the corresponding result obtained in place *Destination*, with the fastest route from *Zone1* to *Zone6*. According to the levels indicated in Fig. 17, transitions *tr1to2*, *tr1to3* and *tr1to5* would be closed, so the only route available is that indicated on place *Destination* in Fig. 18. Notice that this is route **b** in Table 2, but now the average time is different. In Table 2, this route took 11 min and now, due to heavy traffic conditions, it takes 14 min.

Finally, notice that the state graph can now be constructed in an even lower time, because of the restrictions introduced. Some transitions can now be closed, so the number of possible routes can be reduced significantly, and consequently, the graph size can also be reduced. Figure 19 depicts state 6 of the state graph that is obtained with the same simulated conditions of pollutant and traffic levels, which corresponds to the fastest route.

7 Case study: city of Madrid

In this section, the city map CPN model is applied to a real scenario, which is based on the division into districts of Madrid, the capital city of Spain. Madrid consists of 21 districts.[5] In addition, 24 air quality monitoring stations are spread throughout the city (Table 4), but not in all districts. Hourly data gathered from these stations can be read at the URL http://www.mambiente.munimadrid.es/sica/scripts/, where we can also obtain data logs.

Figure 20 shows the 21 districts and the air quality monitoring stations, which are indicated in either yellow or

[5] http://www.madrid.es.

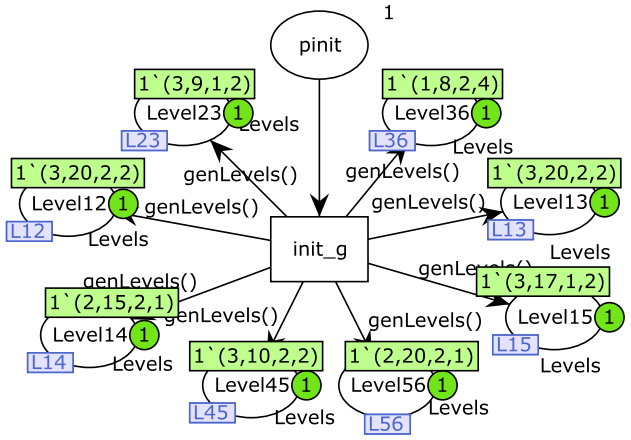


Fig. 17 Generation of pollution and traffic levels

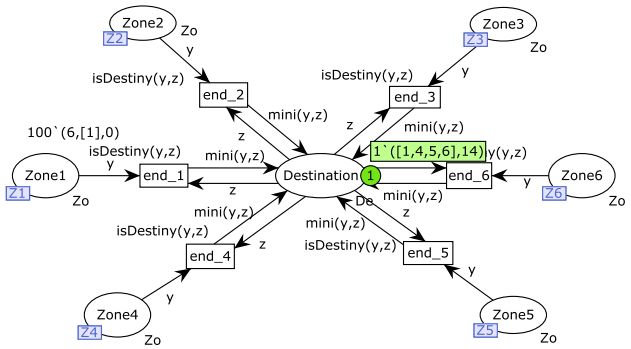


Fig. 18 Destination page after the simulation

green. We decided to join districts 10, 19 and 20—in which there are no measuring stations—to some adjacent districts, in order to establish our city map model, so as to have an air quality monitoring station in each zone. Thus, we consider a map that consists of 18 zones, and in each zone, we have a station that provides us with the air quality information. We call *Zonei* the zone corresponding to district *i* and *Zonei_j* the zone obtained by joining districts *i* and *j*.

From Table 4, we can see that only NO_2 is measured in all stations, so this will be the only pollutant considered in the scenario. In fact, NO_2 is possibly the most significant pollutant in the case of Madrid, due to the peak values it reaches during the episodes of high air pollution. This pollutant is mainly produced by diesel engines, so the city council applies traffic restrictions during these episodes, banning vehicles from driving in the city center.

We have several traffic monitoring stations in each zone, located in the main streets of each district. Thus, we selected one traffic monitoring station in each zone in order to gather the estimated traffic in the zone, so these stations provide us with the general traffic conditions in each zone. The information gathered from these stations can be obtained at https://bit.ly/2Oobzzy. In this case, the

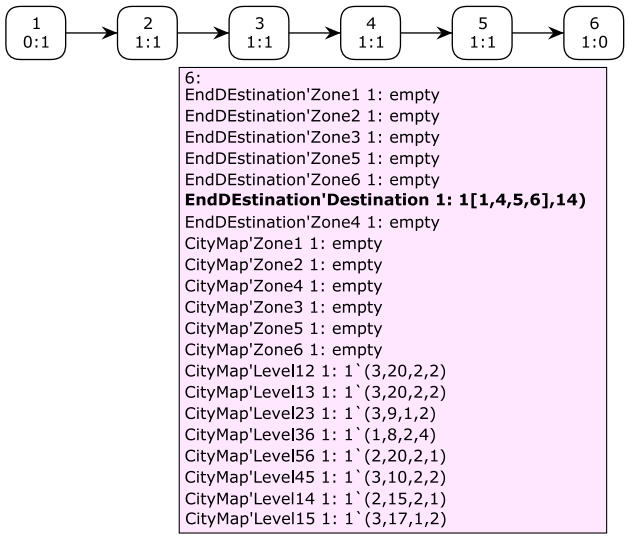


Fig. 19 State space for the generated levels in Fig. 17

information is updated every 5 min and can be rendered using either Google Maps® or Google Earth® apps, as shown in Fig. 21.

We then constructed the city map CPN with 18 zones (see Fig. 22), and the closing conditions were adapted to only one pollutant (NO_2), as follows:

1. The NO_2 pollution level is higher than 3, i.e., Unhealthy, Very Unhealthy or Hazardous.
2. The NO_2 pollution level is Unhealthy for Sensitive Groups (3), but in the last 3 h the increase is always positive between hours and the average increase is higher than 15 ppb.
3. The NO_2 pollution level is Unhealthy for Sensitive Groups (3) and traffic level is either *Saturated* or *Stop* and *go* (levels 3 and 4, respectively).

The pollution and traffic information used to restrict the movements from one zone to another is always based on the destination zone, as shown in Fig. 23, where we can see that transits from *Zone2* to *Zone3* (transition *tr2to3*) use the levels taken from *Zone3*, and conversely for the movements from *Zone3* to *Zone2*.

As an illustration, we have chosen a high pollution scenario that occurred in Madrid on January 23rd, 2018, at 21:00, which is shown in Table 5. Columns H18 to H22 indicate the NO_2 level measured from 18:00 to 22:00, and $P21 and P22$ the average increase in the last 3 h, but only if it is positive, otherwise it is zero. Values greater than 15 for these two last columns are highlighted in bold.

In this table, NO_2 levels are reported in *ppb*, but the information provided by the stations is expressed in $\mu g/m^3$, so a measurement conversion is required. For this conversion, NO_2 pollutant is considered to behave as an ideal gas, so the *ideal gas equation*, $PV = nRT$, is applied for the conversion, taking as temperature T the average value obtained in Madrid on January 23rd, 2018, 283.55 K and taking as pressure (P) the average value in Madrid, 0.9114 atm. In this equation, V is the volume to be obtained, n the number of moles and R the gas constant, $R = 0.082$ atm l/(mol K).

We used these data to obtain the levels indicated in Fig. 24, and then according to our closing conditions, the access to both *Zone5* and *Zone8* is closed at 21:00, because the NO_2 pollution level is Unhealthy for Sensitive Groups (3); in the last 3 h the increase is always positive, and the average increase is higher than 15 ppb. One hour later, at 22:00, access to *Zone5* is open, *Zone8* is still closed, as are *Zone4* and *Zone7*.

With these levels (at 21:00), we obtained the simulation results shown in Fig. 25 for a movement from *Zone1* to *Zone20_21*, using 1000 tokens. This starting value of 1000 tokens was obtained after a number of simulations, with increasing values, until a stable value was obtained. The shortest path obtained for these conditions is indicated by the first field on the token on place *Destination*:

```
1‘([1,2,13,1819,2021],54)
```

which took 54 min (second field).

The results of the state space analysis are shown in Table 6 for the same route used in the simulation. The state space has been constructed for three different scenarios. The first, *No Restrictions*, corresponds to the ideal situation, free flow traffic in which all transits are open, and thus, it produces the largest graph, with the maximum number of both nodes and arcs. The other two scenarios correspond to the data obtained at 21:00 and at 22:00, respectively. In order to explore the state graph produced and thus obtain the shortest path, we use the `SearchAllNodes(pred,eval,start, comb)` function, provided by CPN tools. This traverses all the nodes of the state space, evaluating the `eval` function for the nodes for which predicate `pred` is true. This evaluation starts with the initial value indicated by the expression `start`, and the result is obtained by combining (function `comb`) the value obtained with the previous execution of `eval` and the new value computed for the current node.

Thus, the specific use of this function for our case study is as follows:

```
SearchAllNodes(fn _ => true,fn n =>
hd(Mark.EndDestination’Destination 1 n), start, Shorter)
```

where `start` and `Shorter` are defined as follows:

```
val start = (empty,tmax):De;
fun Shorter(new:De,old:De) = if (#2 new) < (#2  old)
                             then new else old;
```

Table 4 Air quality monitoring stations in Madrid [Type = T (Traffic), U (Urban background), S (Suburban)]

Station	Type	District - number	NO_2	PM10	PM2.5	O_3
Pza. de España	T	Moncloa-Aravaca - 9	Yes			
Esc. Aguirre	T	Salamanca - 4	Yes	Yes	Yes	Yes
Ramón y Cajal	T	Chamartín - 5	Yes			
Cuatro Caminos	T	Chamberí - 7	Yes	Yes	Yes	
Barrio del Pilar	T	Fuencarral - 8	Yes			Yes
Castellana	T	Chamartín - 5	Yes	Yes	Yes	
Pza. Castilla	T	Tetuán - 6	Yes	Yes	Yes	
Pza. del Carmen	U	Centro - 1	Yes			Yes
Mendez Álvaro	U	Arganzuela - 2	Yes	Yes	Yes	
Retiro	U	Retiro - 3	Yes			Yes
Moratalaz	T	Moratalaz - 14	Yes	Yes		
Vallecas	U	Pte. Vallecas - 13	Yes	Yes		
Ens. Vallecas	U	Villa Vallecas - 18	Yes			Yes
Arturo Soria	U	Ciudad Lineal - 15	Yes			Yes
Barajas Pueblo	U	Barajas - 21	Yes			Yes
Urb. Embajada	U	Barajas - 21	Yes	Yes		
Sanchinarro	U	Hortaleza - 16	Yes	Yes		
Tres Olivos	U	Fuencarral-EL Pardo - 8	Yes	Yes		Yes
Juan Carlos I	S	Barajas - 21	Yes			Yes
Casa Campo	S	Moncloa-Aravaca - 9	Yes	Yes	Yes	Yes
El Pardo	S	Fuencarral-El Pardo - 8	Yes			Yes
Fdez. Ladreda	T	Usera - 12	Yes			Yes
Villaverde	U	Villaverde - 17	Yes			Yes
Farolillo	U	Carabanchel -11	Yes	Yes		Yes

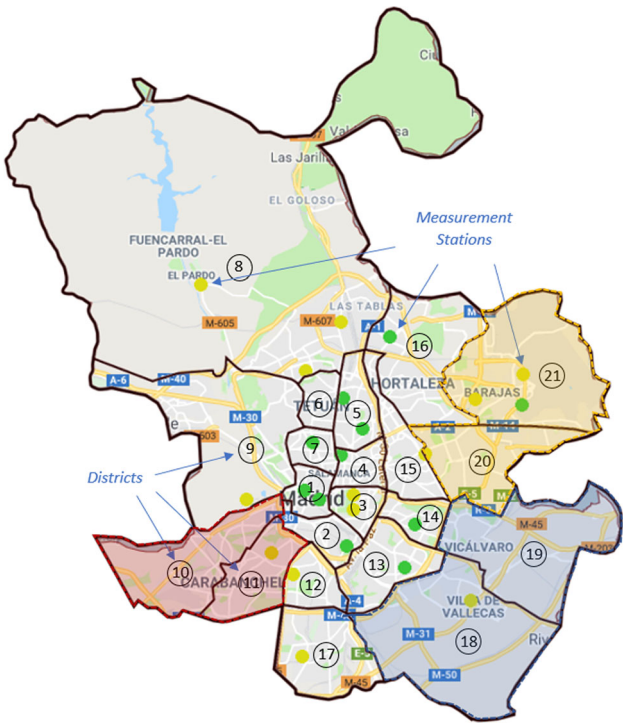


Fig. 20 Districts of Madrid City with Air Quality Measurement stations

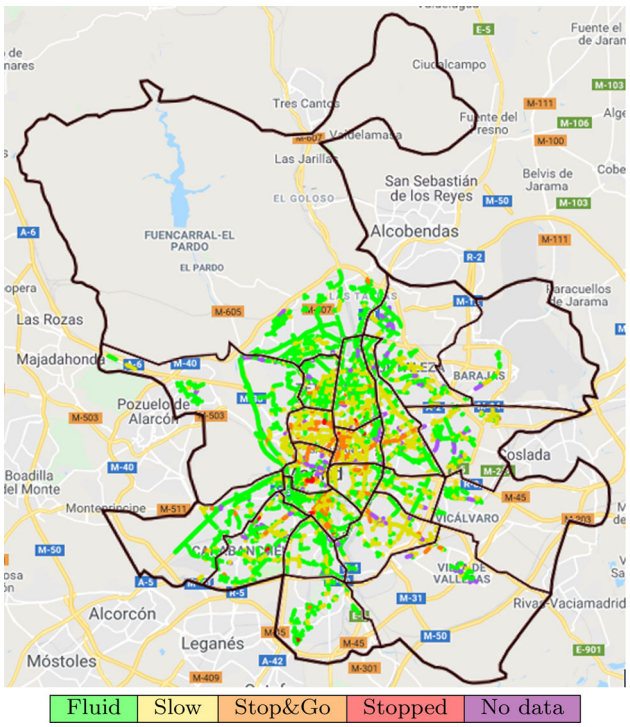


Fig. 21 Madrid districts with traffic levels

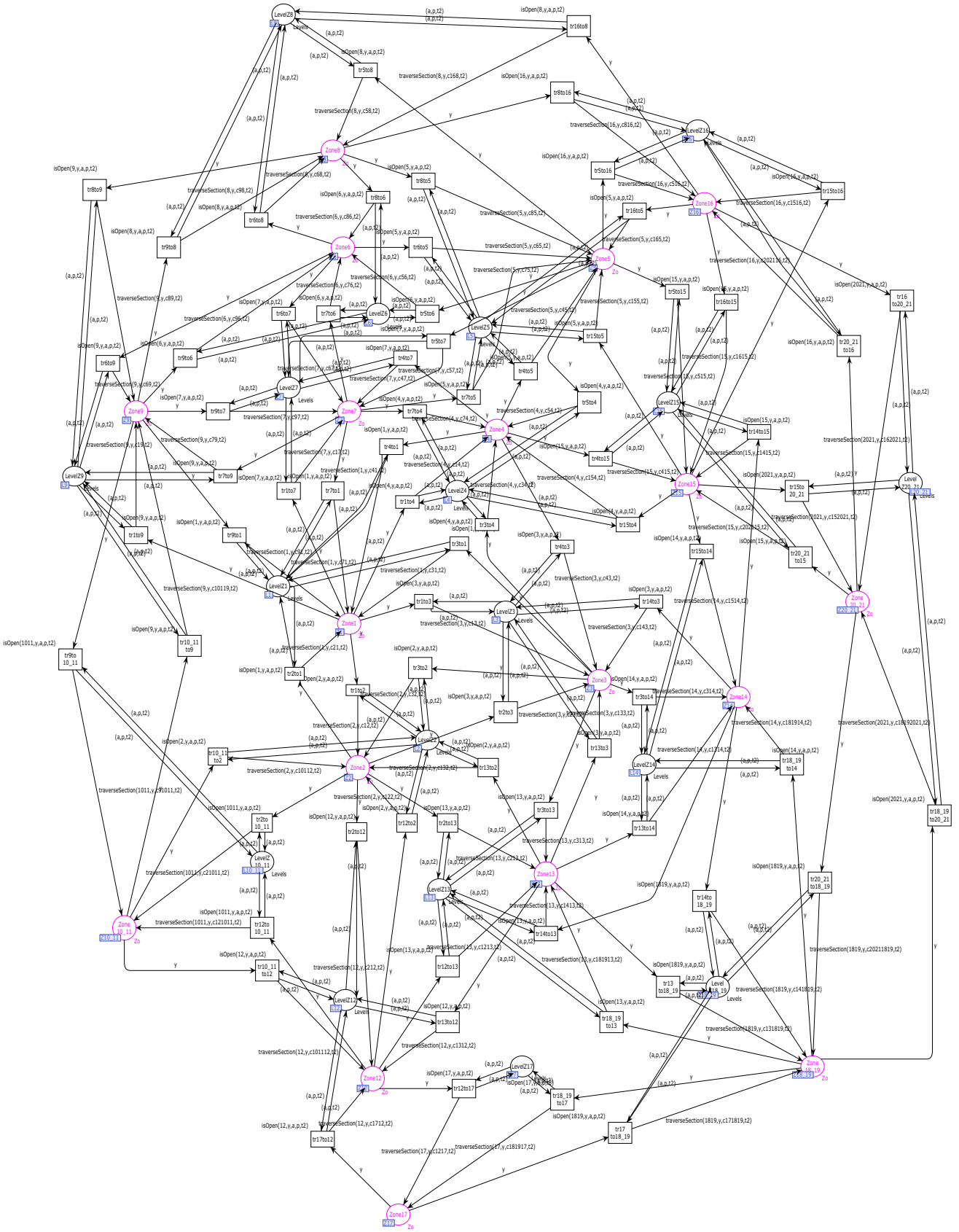


Fig. 22 CPN for the city map of Madrid

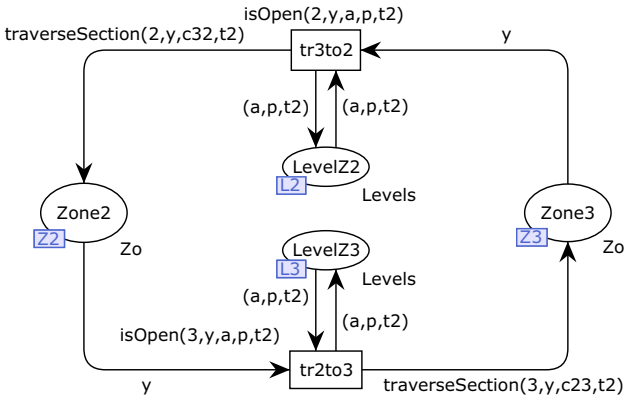


Fig. 23 Transits between *Zone2* and *Zone3*

Function `eval` is then executed for all nodes of the state space in order to obtain the token in the `Destination` place of page `EndDestination`, comparing the second field (time spent) with the previous value so as to obtain the shortest one. The resulting value after evaluating this expression for the 21:00 scenario is 1‘([1, 2, 13, 1819, 2021], 54), which coincides with the value that was obtained by simulation.

Two strategies have also been applied to reduce the exploration. The first, called *Branching Pruning*, uses the *Branching and Stop* options of CPN tools to prune the exploration when a path has a cost greater than another one already computed. In this case, the exploration following that path is stopped, so a significant reduction is obtained, as we can see from the table in the three scenarios. The predicate used for the *Branching Pruning* is the following:

```
fn n => if (Mark.EndDestination'Zone1 1 n <> empty
   andalso Mark.EndDestination'Destination 1 n <> empty)
then  #3(hd (Mark.EndDestination'Zone1 1 n)) < VALUE
else if (Mark.EndDestination'Zone2 1 n <> empty
   andalso Mark.EndDestination'Destination 1 n <> empty)
then  #3(hd (Mark.EndDestination'Zone2 1 n)) < VALUE
        .
        .
        .
else if (Mark.EndDestination'Zonei 1 n <> empty
   andalso Mark.EndDestination'Destination 1 n <> empty)
then  #3(hd (Mark.EndDestination'Zonei 1 n)) < VALUE
```

Where `VALUE` is the expression:

```
SearchAllNodes(fn _ => true,fn n =>
         #2 (hd(Mark.EndDestination'Destination 1 n)),
                                        tmax, Int.min)
```

In this predicate, every *if* statement checks whether the `Zonei` and `Destination` places contain any tokens and every *then* clause returns whether the travel time of the token at `Zonei` is lower than those already explored that were obtained with the expression `VALUE`. Thus, the state space exploration of a certain branch is **pruned** if the travel time obtained is higher to this value.

In addition, we propose the use of simulation results as a way to also prune the exploration, because we stop all branches for which the route has a cost greater than that obtained from simulations. This strategy is called *Simulation Pruning*, and we can see from the table that it provides the best results in the three scenarios, although in general this will depend on the quality of the results obtained by simulation.

Table 5 Data gathered from Madrid, 23rd January 2018

Station	ID	District	Traffic	H18	H19	H20	H21	H22	P21	P22
Pza. del Carmen	28079035	1	3	46.56	52.1	57.09	58.19	48.2	6	0
Méndez Álvaro	28079047	2	3	41.57	61	84.8	73.16	56.5	0	0
Retiro	28079049	3	3	37.13	35.5	39.91	46	48.8	0	6
Escuelas Aguirre	28079008	4	4	53.76	69.8	90.34	94.77	102	**19**	**18**
Av. Ramón y Cajal	28079011	5	2	37.69	71.5	92.56	131.9	106	**43**	0
Pza. Castilla	28079050	6	3	43.23	64.3	70.39	100.3	89.2	**26**	0
Cuatro Caminos	28079038	7	4	53.21	72.6	90.34	95.33	109	**21**	**18**
Barrio del Pilar	28079039	8	1	41.57	62.6	103.1	115.8	130	**32**	**36**
Pza.de España	28079004	9	3	43.23	52.1	64.29	59.86	49.9	0	0
C/ Farolillo	28079018	11	1	48.22	68.2	71.5	63.18	61	0	0
Pza. Fdez. Ladreda	28079056	12	1	59.86	101	125.3	63.18	63.7	0	0
Vallecas	28079040	13	2	41.01	53.8	63.74	74.82	78.2	**16**	11
Moratalaz	28079036	14	3	35.47	47.7	73.71	94.77	67.1	**24**	0
Arturo Soria	28079016	15	2	25.49	46	59.86	69.28	65.4	**21**	0
Sanchinarro	28079057	16	1	32.15	66.5	81.47	102	95.9	**35**	0
Villaverde Alto	28079017	17	1	44.89	73.7	85.35	87.02	74.8	**24**	0
Ensanche Vallecas	28079054	18	1	37.69	67.1	100.9	97.55	105	0	0
Urb. Embajada	28079055	21	1	30.48	52.1	67.06	67.06	70.4	0	0

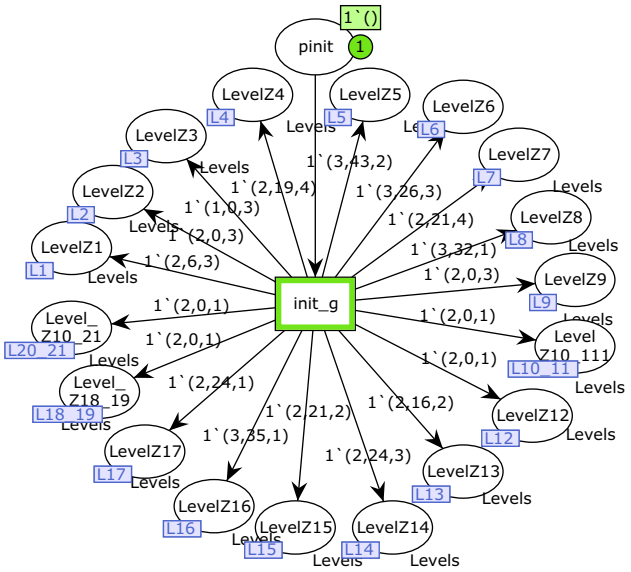


Fig. 24 Levels in Madrid City at 21:00 hours, January 23rd, 2018

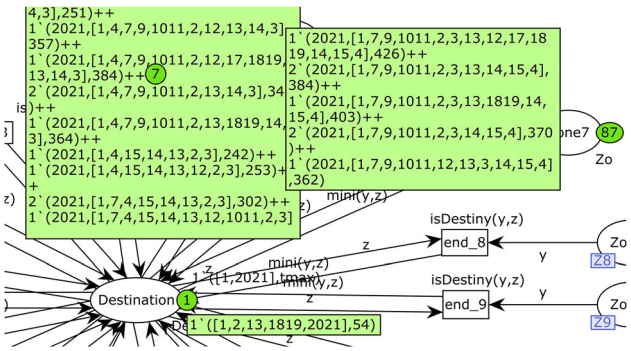


Fig. 25 Simulation results from Zone1 to Zone20_21

Table 6 State space report results

	Nodes	Arcs	Secs	Status
No restrictions				
Complete state space	209,897	213,535	5551	Full
Branching pruning	396	399	0	Partial
Simulation pruning	392	395	0	Partial
21:00 hours				
Complete state space	4300	4393	2	Full
Branching pruning	183	186	0	Partial
Simulation Pruning	90	91	0	Partial
22:00 hours				
Complete state space	2902	2945	1	Full
Branching pruning	148	150	0	Partial
Simulation pruning	91	92	0	Partial

In fact, from the analysis of the graph we concluded that the best route in all scenarios was that obtained by simulations, since none of the routes in the graph obtained by using the *Simulation Pruning* strategy could reach its destination (all were worse and so were stopped). The predicate for the *Simulation Pruning* is similar to the previous predicate but substituting the `VALUE` expression with the values obtained by simulation (40, 54 and 54, respectively).

7.1 Scalability

We selected a total of 18 zones for a large city like Madrid, since in general the areas to be closed (or simply restricted) cover one or even several districts. This is because pollution is not a confined problem and the affected area can be very wide. In the case of Madrid, the entire city center is usually affected by traffic regulations when there is an episode of high pollution. Thus, we do not expect to have a city with hundreds of zones. Thus, the scalability analysis is performed on two scenarios: a city map with 36 zones (twice the size of the original city map) and another with 54 zones (three times the original city map size).

The CPN was constructed by replicating the city map of Madrid and establishing connections between these CPNs in order to obtain these scenarios. In this way, the only interconnections considered are between *Zone18_19* from the first copy to *Zone9* of the second copy and from *Zone20_21* of the first copy to *Zone8* of the second copy, taking 12 and 10 min, respectively. The same strategy is applied in the second scenario between the second and third copy.

Table 7 shows the results for the two scenarios considered—from *Zone1* of the first copy to *Zone20_21* of the second copy (36 zones) or to *Zone20_21* of the third copy (54 zones)—with "*No Restrictions*" (all connections open), so as to obtain the largest state space graph. Notice that the state space graph is constructed by using one single starting token on the *Zone1* place. In neither case could the complete graph be constructed, due to the state space explosion. For the city map of 36 zones, the *Branching Pruning* strategy took 66 s for completion, and *Simulation Pruning* 68 s.

The simulation with 5, 000 tokens yielded the following result:

```
1‘([1,4,15,2021,108,116,12021],82)
```

We then took 82 as the value used for the *Simulation Pruning* strategy, thus obtaining the optimal solution in both cases:

```
1‘([1,2,13,1819,109,106,105,116,12021],80)
```

Therefore, the movement from *Zone1* of the first copy to *Zone20_21* of the second copy took 80 min for the best route.

In contrast, for the city map of 54 zones, none of these techniques yielded any results with little response latency, so we applied a new *Simulation Pruning* strategy by splitting the route into legs.

This strategy works as follows. First, a simulation is run with a relatively high number of tokens, e.g., 5, 000 tokens, in order to obtain a suboptimal itinerary between *Zone1* of the first copy to *Zone20_21* of the third copy. The suboptimal solution obtained was:

```
1‘([1,3,13,1819,
          109,108,105,116,12021,
                    208,216,22021],146)
```

where the numbers in the second and third lines correspond to zones in the second and third copies, respectively. Then, we applied a simulation with 5, 000 tokens restricted to the first copy for a movement from *Zone1* to its connections with the second copy (*Zone18_19* or *Zone 20_21*), obtaining 40 min as the best result. In the same way, we applied a new simulation to obtain a suboptimal value for a movement from *Zone1* of the first copy to *Zone18_19* or *Zone 20_21* of the second copy. In this case, the value obtained was 90 min. Finally, *Simulation Pruning* strategy was applied, considering the following threshold values in each copy: 41, 91 and 147, respectively. These values are higher than the values obtained in the simulation to avoid a premature ending in the state space exploration in the case that these values were optimal. The results are shown in the last row of Table 7, and the optimal solution was:

```
1‘([1,2,13,1819,
        109,106,105,116,12021,
                  208,216,22021],122)
```

which takes 122 min for a movement from *Zone1* of the first copy to *Zone20_21* of the third copy.

8 Related work

In recent years, several works have proposed ITS models based on PN formalisms. In order to enhance them with the functionalities provided by the CEP technology, we proposed an unprecedented ITS model [22], which integrates both PNs and the CEP technology to analyze traffic regulations only based on the CO pollutant level imposed by the EPA.

Table 7 Scalability results

	Nodes	Arcs	Secs	Status
(36 zones)				
Branching pruning	12,959	12,958	66	Partial
Simulation pruning	13,272	13,271	68	Partial
(54 zones)				
Simulation pruning	17,752	17,751	25	Partial

Regarding other works on modeling of ITS systems by using PNs, Cavone et al. [23] present a survey on freight logistics and transportation systems based on PN formalisms together with applications to analysis, simulation, optimization and control. Junior et al. [24] propose an analytical model based on the Stochastic PN (SPN) theory for evaluating Vehicular Ad-Hoc Networks (VANETs) infrastructures, where expolynomial distributions are used to represent roadside unit service rates. They study the overall system performance taking into account parameters such as vehicular density, message frequency and RSU (RoadSide Unit) radius. Qi et al. [25] make use of deterministic and stochastic PNs to design an emergency traffic-light control system for intersections prone to accidents. Reachability analysis techniques are then used to prevent deadlocks and livelocks. Júlvez and Boel [26] propose a dynamic model based on continuous PNs to model the macroscopic behavior of traffic systems. The proposed traffic model provides a predictive control strategy on traffic systems taking into account that traffic conditions may vary quickly. The authors focus on the minimization of the total delay (waiting time) of the cars in the system. Hübner et al. [27] propose a vehicle formation model for traffic optimization by means of a consensus algorithm and a condition event PN, which is used to model the topology of the vehicles' positions. The goal formation is characterized by maximum vehicle density and limited interactions. Riouali et al. [28] use Generalized non-deterministic batch Petri Nets (GNBPN), which are an extension of hybrid PNs. They model discrete and continuous aspects of traffic flow dynamics, taking into account state dependencies based on external rules, such as stop sign or priority roads. ČapkoviČ [29] uses three different models of PNs to model segments of a transport network, namely P/T Petri Nets, Timed Petri Nets and Hybrid Petri Nets. P/T Petri Nets are used to find the safe and unambiguous structure of the traffic-light controller. Then, Timed Petri Nets are used to analyze the time relations between the traffic lights, and finally, Hybrid Petri Nets are used to find flows of vehicles within the bounds of possibility determined by the traffic lights. Bonnefoi et al. [30] propose a specification methodology based on a set of UML diagrams to generate an analyzable PN formal model of an ITS. The system requirements are then expressed as LTL or CTL properties, which are verified by using a PN model checker. Aitouche et al. [31] present a multiagent model using CPNs for traffic regulation of an automated highway. Their goal is to find solutions for the problem of congestion in Automated Highway Systems (AHS), taking into account the departure and arrival time of vehicles, ensuring their correct routing.

Huang et al. [32] use Synchronized Timed Petri Nets in a methodology to design and analyze an urban traffic network system in a modular way, showing the clear presentation and readability of such design. Qi et al. [33] use Timed Petri Nets to design a two-level strategy at signalized intersections for preventing incident-based urban traffic congestion by adopting additional traffic warning lights.

9 Conclusions and future work

In this paper, we propose an ITS model for traffic control considering both air pollutant and traffic levels with the aim of alleviating not only pollution but also other traffic problems. The proposed model conforms to the three aspects that an ITS should satisfy: (1) consistent with traffic flow, (2) flexible to characterize a dynamic flow and (3) simple, but rich enough to allow us to draw conclusions about traffic regulations.

Air quality conditions and traffic flow data are assumed to be determined by a set of sensor stations. The CEP technology is then used to process the information gathered from these sensors and determine the AQI and traffic levels. Specifically, we use the MEdit4CEP-CPN tool to model the event patterns regarding the $PM_{2.5}$ pollutant as well as the traffic level event patterns. A CPN modeling of a city map is then defined to compute the optimal available routes taking into account that some connections may be closed due to air quality or traffic conditions. In addition, an important feature of the CPN model is that transit times between zones are computed taking into account that traffic density is likely to affect them.

The analysis techniques of CPN tools are used to obtain these optimal routes. One of these techniques is state exploration based on constructing the state space graph, which allows us to obtain all possible routes and thus obtain the optimal one. However, the state space graph can generally be quite large, and in some cases it could even be impossible to generate. In these cases, simulation techniques are to be applied, which quickly provide suboptimal solutions. The simulation technique applied in this work is based on generating a number of identical tokens on the starting zone, so as to cover as many routes as possible, and return the fastest one as the suboptimal solution. State space exploration can benefit from the branching and stop options of CPN tools so as not to construct all the state space, pruning the branches that have a higher cost than others previously computed. We apply this technique, showing the benefits it provides in both time and graph size. We also propose a combination of both simulation and state exploration, using the results obtained by simulation in order to apply the branching and stop exploration.

As future work, the CPN model could be enriched by improving the initial transit times between zones, taking into account, for instance, the size of the regions and the starting and ending points inside each region. Other key information to compute the routes could be to consider traffic-light information or accidents that could cause delays or closing connections.

Other aspects could also be considered, such as closing to traffic on the basis of type of fuel and emissions of vehicles.

A final step would be to integrate this CEP-based solution with an Enterprise Service Bus (ESB) to test this approach in a real scenario, in which sensing data will be produced by heterogeneous and ubiquitous sources.

NOTE: All the CPN models presented in the paper, the reports obtained, and the predicates used for the application of branching and stop options are available via the link: https://doi.org/10.17632/cbjxbhzn43.1.

Acknowledgements The authors would like to thank Prof. Dr. Edelmira Valero, member of the Physical Chemistry Department at the University of Castilla-La Mancha, for her helpful cooperation in the matters related to air pollutants and conversions of units of measure. Boubeta-Puig would like to thank the Real-Time and Concurrent Systems Research Group for their hospitality when visiting them at the University of Castilla-La Mancha, Spain, where part of this work was developed.

Funding This study was funded in part by the Spanish Ministry of Science and Innovation and the European Union FEDER Funds under Grants TIN2015-65845-C3-2-R, TIN2015-65845-C3-3-R and TIN2016-81978-REDT, and also by the JCCM regional project SBPLY/17/180501/000276, which is also co-financed by the European Union FEDER Funds.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Picone M, Busanelli S, Amoretti M, Zanichelli F, Ferrari G (2015) Advanced technologies for intelligent transportation systems. Springer, New York
2. Bekiaris E, Nakanishi J (2004) Economic impacts of intelligent transportation systems, vol 8. Elsevier, Amsterdam
3. Xu W, Zhou H, Cheng N, Lyu F, Shi W, Chen J, Shen XM (2016) Internet of vehicles in big data era. IEEE/-CAA J Autom Sin 5:19–35

4. World Health Organization: Health Risk Assessment of Air Pollution. General Principles (2016). http://www.euro.who.int/en/health-topics/environment-and-health/air-quality/publications/2016/health-risk-assessment-of-air-pollution.-general-principles-2016. Accessed 16 Oct 2018
5. Luckham D (2012) Event processing for business: organizing the real-time enterprise. Wiley, Hoboken
6. Boubeta-Puig J, Díaz G, Macià H, Valero V, Ortiz G (2017) MEdit4CEP-CPN: an approach for complex event processing modeling by prioritized Colored Petri Nets. Inf. Syst. https://doi.org/10.1016/j.is.2017.11.005 (**in press**)
7. Boubeta-Puig J, Ortiz G, Medina-Bulo I (2015) MEdit4CEP: a model-driven solution for real-time decision making in SOA 2.0. Knowl Based Syst 89:97–112. https://doi.org/10.1016/j.knosys.2015.06.021
8. Peterson JL (1981) Petri net theory and the modeling of systems. Prentice Hall PTR, Upper Saddle River
9. Air Quality in Europe (2017) Technical report no 13/2017, European Environment Agency (EEA). Publications Office of the European Union, Luxembourg
10. The Economic Consequences of Outdoor Air Pollution (2016) Technical report, Organisation for Economic Co-operation and Development (OECD). OECD Publishing, Paris. http://www.oecd-ilibrary.org/environment/the-economic-consequences-of-outdoor-air-pollution_9789264257474-en. Accessed 16 Oct 2018
11. EPA: Technical Assistance Document for the Reporting of Daily Air Quality—the Air Quality Index (AQI). Technical report EPA-454/B-16-002, U.S. Environmental Protection Agency, North Carolina, US (2016). https://nepis.epa.gov/Exe/ZyPURL.cgi?Dockey=P100FD3G.TXT. Accessed 16 Oct 2018
12. Yuan C, Ng E, Norford LK (2014) Improving air quality in high-density cities by understanding the relationship between air pollutant dispersion and urban morphologies. Build Environ 71:245–258
13. Borge R, de Miguel I, de la Paz D, Lumbreras J, Pérez J, Rodríguez E (2012) Comparison of road traffic emission models in Madrid (Spain). Atmos Environ 62:461–471
14. Etzion O, Niblett P (2010) Event processing in action, 1st edn. Manning Publications Co., Greenwich
15. Cugola G, Margara A (2012) Processing flows of information: from data stream to complex event processing. ACM Comput Surv 44(3):15:1–15:62
16. Esper—Complex Event Processing. http://www.espertech.com/esper/. Aug 2018
17. CPN tools home page. http://www.cpntools.org/. Aug 2018
18. van der Aalst WMP, Stahl C, Westergaard M (2013) Strategies for modeling complex processes using Colored Petri Nets. Trans Petri Nets Other Models Concurr 7:6–55. https://doi.org/10.1007/978-3-642-38143-0_2
19. Jensen K, Kristensen LM (2009) Coloured Petri Nets: modelling and validation of concurrent systems, 1st edn. Springer, Berlin
20. Li L, Lv Y, Wang FY (2016) Traffic signal timing via deep reinforcement learning. IEEE/CAA J Autom Sin 3(3):247–254
21. Sustainable mobility ordinance (2018) Boletín Oficial de la Comunidad de Madrid, vol 253. Madrid City Council, pp 130–251. http://www.bocm.es/boletin/CM_Orden_BOCM/2018/10/23/BOCM-20181023-36.PDF
22. Díaz G, Macià H, Valero V, Cuartero F (2017) Intelligent transportation system to control air pollution in cities using Complex Event Processing and Colored Petri Nets. In: Advances in computational intelligence. Lecture notes in computer science. Springer, pp 415–426. https://doi.org/10.1007/978-3-319-59147-6_36
23. Cavone G, Dotoli M, Seatzu C (2018) A survey on petri net models for freight logistics and transportation systems. IEEE Trans Intell Transp Syst 19(6):1795–1813. https://doi.org/10.1109/TITS.2017.2737788
24. Junior AL, Matos R, Silva B, Maciel P (2017) Expolynomial modelling for supporting VANET infrastructure planning. In: 2017 IEEE 22nd Pacific rim international symposium on dependable computing (PRDC), pp 86–91. https://doi.org/10.1109/PRDC.2017.20
25. Qi L, Zhou M, Luan W (2016) Emergency traffic-light control system design for intersections subject to accidents. IEEE Trans. Intell. Transp. Syst. 17(1):170–183. https://doi.org/10.1109/TITS.2015.2466073
26. Júlvez J, Boel RK (2010) A continuous petri net approach for model predictive control of traffic systems. Trans Syst Man Cybern Part A 40(4):686–697. https://doi.org/10.1109/TSMCA.2010.2041448
27. Hübner M, Lück T, Schnieder E (2009) Cooperative control of multi-vehicle-formations in road traffic by means of consensus algorithm and petri nets. IFAC Proc Vol 42(15):328–333. https://doi.org/10.3182/20090902-3-US-2007.0016
28. Riouali Y, Benhlima L, Bah S (2016) Petri net extension for traffic road modelling. Int J Sci Eng Res 7(11):282–299
29. Čapkovič F (2015) Petri net-based modelling and simulation of transport network segments. In: Propagation phenomena in real world networks. Intelligent systems reference library, vol 85. Springer, Cham, pp 135–154. https://doi.org/10.1007/978-3-319-15916-4_6
30. Bonnefoi F, Hillah LM, Kordon F, Renault X (2007) Design, modeling and analysis of ITS using UML and petri nets. In: 2007 IEEE intelligent transportation systems conference, pp 314–319. https://doi.org/10.1109/ITSC.2007.4357718
31. Aitouche A, Hayat S (2003) Multiagent model using Coloured Petri Nets for the regulation traffic of an automated highway. In: Proceedings of the 2003 IEEE international conference on intelligent transportation systems, vol 1, pp 37–42. https://doi.org/10.1109/ITSC.2003.1251916
32. Huang YS, Weng YS, Zhou M (2014) Modular design of urban traffic-light control systems based on synchronized timed petri nets. IEEE Trans Intell Transp Syst 15:530–539
33. Qi L, Zhou M, Luan W (2018) A two-level traffic light control strategy for preventing incident-based urban traffic congestion. IEEE Trans Intell Transp Syst 19(1):13–24