Distributed Planning and Model Learning for Urban Traffic Control

Alberto Pozanco **Susana Fernández** **Daniel Borrajo**
Departamento de Informática, Universidad Carlos III de Madrid
Avda. de la Universidad, 30. 28911 Leganés (Madrid). Spain
apozanco@pa.uc3m.es, sfarregu@inf.uc3m.es, dborrajo@ia.uc3m.es

Abstract

Urban Traffic Control is a key problem for most big cities. Current approaches to handle the city traffic rely on controlling traffic lights. The systems in operation range from static control of traffic light phases to adaptive systems based on numeric models and traffic sensors. Recently, some planning-based approaches have also been proposed. We have identified two main difficulties to the wide use of planning techniques in this domain: generating the control models is a difficult task; and some algorithms scale poorly. In this paper we present APTC, a control system based on Automated Planning, that successfully overcomes these two problems. It combines techniques that continuously: learn an accurate planning model; and also divide the city for distributed reasoning in order to scale to large city networks. Experimental results show that APTC outperforms static approaches as well as other planning-based systems. We also show that the combination of both approaches improves over using only one of them.

Introduction

In the last decades the world's population has grown steadily, becoming more urbanised over the years. This growth consequently increases the public transport demand as well as the number of cars and vehicle movements. However, traffic infrastructures do not grow at the same rate, so implementing efficient Urban Traffic Control (UTC) systems is increasingly important. Hence, traffic management can influence the entire city dynamics, causing significant damages to the population, ranging from unnecessary time and fuel consumption up to deteriorating citizen's health. The traffic control task is a difficult one, since it involves stochastic behavior by independent agents (drivers) and plenty of unforeseen events that can affect the transportation network, such as maintenance, accidents, weather or sports events.

Most current systems control the city traffic using macroscopic approaches (Treiber and Kesting 2013) that model traffic at the flow level rather than taking into account isolated cars. They usually set traffic lights programs, that are defined in terms of three parameters: split and cycle, which refer to the amount of green and red time allocated to each traffic light; and offset, that represents the difference between the start of green time at two consecutive intersections. An appropriate offsets' setting at various connected traffic lights generates "green waves" that allow vehicles not to stop in several consecutive junctions. There are many known algorithms to define those programs, ranging from early static off-line approaches that are still in use in many cities, to most recent adaptive approaches that change the programs according to the network's state (Papageorgiou et al. 2007; Hamilton et al. 2013). An example of a successful adaptive approach is the SCOOT system, a commercial product that uses information coming from traffic sensors to feed numerical models (Bretherton, Wood, and Bowen 1998). A weak point of these systems is that they cannot deal well with dynamic incidents (Vallati et al. 2016). Also, their models are usually difficult to maintain.

Recently, some Artificial Intelligence (AI) approaches have emerged, using diverse techniques: neural networks (Box and Waterson 2012), reinforcement learning (Jin and Ma 2017); or scheduling techniques (Xie, Smith, and Barlow 2012). Automated Planning (AP) has also been recently shown to perform well in this kind of tasks (Cenamor et al. 2014; Gulič, Olivares, and Borrajo 2016; Vallati et al. 2016). The main advantage of using AP is that the problem can be modeled using a declarative language in combination to powerful reasoning engines. Thus, traffic engineers can easily include or modify new actions, sensor information or metrics to adapt the model to evolving traffic conditions.

In general, previous techniques and particularly the ones based on AP have two main drawbacks. Firstly, given the stochastic nature of the control task, as well as the amount of different ways to control traffic lights, properly modeling the planning task requires some knowledge engineering effort. Moreover, in most cases, the defined model does not perfectly fit the real scenario, which affects the system's behavior. Also, most of these models assume all junctions share the same behavior within a city and across cities. Secondly, these approaches scale poorly and can not be implemented in large areas or cities with numerous streets and traffic lights.

In this paper we present APTC (Automated Planning for Traffic Control), a system based on AP to perform UTC. The goal of APTC is to overcome the previously mentioned two problems. First, we propose to automatically update the planning domain model through continuous incremental learning. The automatic generation of planning domains in stochastic environments has been previously studied (García-Martínez and Borrajo 2000; Pasula, Zettle-

moyer, and Kaelbling 2007; Jiménez et al. 2012; Jiménez, Fernández, and Borrajo 2013; Martínez et al. 2016; Mourao 2014). However, these domain-independent approaches are not adequate when the learning tasks involve actions that use many parameters as it is the case of UTC actions. Therefore we propose a domain-dependent model updating approach. The learning technique monitors the observed states at each junction and automatically generates actions that match the junction's dynamics. This allows APTC to quickly adapt to the city traffic behavior and its changes, generating better planning domains requiring less model engineering work.

Second, we propose to use distributed planning by dividing the city into a set of areas. Given that it is difficult to define those areas "a priori", APTC identifies the most important junctions and divides the city according to the traffic flows that dynamically arise or disappear over time. This city splitting criteria not also leads APTC to better scalability. By keeping the junctions involved in a flow in the same planning problem, APTC can automatically generate "green waves". The system generates them by setting the traffic lights in such a way that they allow the vehicles to quickly traverse the junctions involved in the traffic flows. A planning problem is generated in each area and they are solved asynchronously. The resulting plans are concatenated and executed to control the traffic lights. APTC can be seen as an instance of a fully autonomic (autonomous) system (Huebscher and McCann 2008), given that it incorporates many self-* properties, as self-monitoring (continuous observation), self-diagnosis (undesired behavior detection), self-optimization (planning), self-healing (execution of traffic control actions) and self-adaptation (learning).

Planning Models for UTC

We propose to use AP to solve traffic control tasks. From all the different kinds of available planning techniques, we will use those that take as input an explicit model described in the standard PDDL language (Planning Domain Description Language) (Fox and Long 2003). These planning techniques take as input a planning task and return a plan that solves it.

Planning Models

A single-agent STRIPS planning task can be formally defined as a tuple $\Pi = \{F, A, I, G\}$, where F is a set of propositions, A is a set of instantiated actions, $I \subseteq F$ is an initial state, and $G \subseteq F$ is a set of goals. Each action $a \in A$ is described by a set of preconditions (pre(a)), that represent propositions that must be true (or false for negative preconditions) in a state to execute the action and a set of effects (eff(a)), propositions that are expected to be true (add(a) effects) or false (del(a) effects) after execution of the action. The application of an action a in a state s is defined by a function γ, such that $\gamma(s, a) = (s \setminus \text{del}(a)) \cup \text{add}(a)$ if pre(a)$\subseteq$ s and s otherwise (it cannot be applied). Planners should generate as output a sequence of actions, called a plan, $\pi = (a_1, \ldots, a_n)$ such that if applied in order from the initial state I would result in a state s_n, where the goals are true, $G \subseteq s_n$. Under this definition, A and F are fully grounded propositions. To alleviate the definition of planning tasks, the AP community has defined PDDL, a high-level language that allows planning users to easily define these tasks. It is based on predicate logic, and requires the definition of two files: domain and problem. The domain model D contains the definition of predicates for representing sets of propositions and the actions that agents can perform. The problem P describes the particular task instance to be solved; i.e., the objects involved, the initial state and the set of goals to achieve.

Modeling UTC with PDDL

There have been different approaches to model UTC from an AP point of view, but all of them rely on acting over the traffic lights. Vallati *et al.* 2016 propose a PDDL+ formulation (Fox and Long 2006). PDDL+ is an extension of PDDL to model mixed discrete-continuous domains. This approach switches the traffic lights for a certain amount of time depending on the queues of vehicles on the streets entering the junction.

In parallel, and using a simpler PDDL domain, Gulić *et al.* modeled UTC taking into account street's density levels instead of flows and queues of cars in their IAS system (Gulìc, Olivares, and Borrajo 2016). Actions are executed over the city network only when a high density level is detected at any street. This is done for a fixed time and then the system monitors if the congestion has been solved, i.e., the density is low in all streets, returning to the default program. This planning model assumes the world is deterministic and the agent has full observability. But UTC does not follow these premises, since the actions have stochastic outcomes and the agents have partial observability. In order to deal with uncertainty, they follow a simple and popular approach; reasoning (planning) with a deterministic world model and when execution of some action fails (the congestions are not solved), the agent replans (Yoon, Fern, and Givan 2007). Besides, as in robotics, this approach usually employs a simplified model to solve high-level deliberative problem solving. And there is a low-level reasoning model that takes care of some of the complexities of dealing with numeric quantities and continuous processes. In their case, the low-level model translates high-level actions into each crossing's traffic lights for specific amounts of time. In this paper, we will take the UTC model used in IAS as a baseline. It presents two advantages over using PDDL+: there are many more planners that can work with PDDL; and discrete models are usually easier to define and learn. The potential disadvantage would be that the IAS PDDL models are not as accurate as the PDDL+ ones, given that they do not deal with continuous numeric models. However, we can alleviate this problem by the low-level (simple) behavior in this case where the relevant parameters to traffic lights control are set appropriately given a high-level action. And, also, by our learning mechanism later described.

The first step when generating a planning domain is to determine the predicates and the actions to use. Each high-level action in IAS controls all the traffic lights of a junction at once. As an example, in a four-way junction there are four incoming streets and another four outgoing streets (each with a finite number of lanes) and each incoming street

```
(:action one-high
 :parameters (?c - junction ?sin1 - street ...
              ?sout1 - street ...)
 :precondition (and (goes-into ?sin1 ?c)
                    (in-front-of ?sin1 ?sin3)
                    ...
                    (goes-out ?sout4 ?c)
                    (not (= ?sin1 ?sin2))
                    ...
                    (densityLevel ?sin1 high)
                    (densityLevel ?sin3 low)
                    ...)
 :effect (and (densityLevel ?sin1 low)
              (not (densityLevel ?sin1 high))
              (densityLevel ?sin3 low)
              (not (densityLevel ?sin3 high))
              (densityLevel ?sout2 low)
              (not (densityLevel ?sout2 high))))
```

Figure 1: Part of an example description of a PDDL action that sets one traffic light of a junction to green.

is controlled by one traffic light. The actions that can be applied at this specific type of junctions are limited: you can set to green one of the four traffic lights separatedly, or set to green two of them if they are in front of each other. The high-level action that sets to green a traffic light is translated into several low-level actions to set to red the other traffic lights in the junction.

Since we want to model how to control traffic lights, predicates allow us to represent the current state of the junction. We use most of the predicates defined in IAS. There are static predicates that reflect the city network composition such as `goes-into` and `goes-out` to indicate whether a street enters or leaves a junction; and dynamic predicates such as `densityLevel` that indicates the congestion level of a given street.[1] Since sensors return numeric values for the density, we applied a discretization step for its values, using a threshold. We use the same two density values defined by Gulić *et al.*; `high` for density values higher than 0.35 and `low` otherwise.

In UTC, actions should decide how to control the traffic lights. In our case, we will use again the same approach introduced by IAS that sets the traffic lights to green when it decides that the default program is performing badly. So, the preconditions of each action check if the densities of some street sections are high (dynamic predicates) as well as some static preconditions (network structure around each junction). If the action is executed, the effects describe the expected changes in the new state. Figure 1 shows the definition of an action in which a traffic light is set to green.

Problems are mainly composed of a set of objects, an initial state and a set of goals. In our case, the objects are the streets and the junctions. The initial state would be composed of: the static part of the city i.e., the connections between the streets and the geometry of the junctions; and the dynamic part made of the initial density levels of the streets. The goal would be to have low density in all the streets. A potential planning problem would be as shown in Figure 2.

[1]This value is provided by the simulator we use, and it would be generated by street sensors in a real scenario.

```
(define (problem traffic1) (:domain traffic)
  (:objects sin1 ... sout7 - street
            j1 j2 - junction)
  (:init (goes-into sin1 j1)
         (in-front-of sin1 sin3)
         (densityLevel sin1 high)
         (densityLevel sin2 low)...)
  (:goal (and (densityLevel sin1 low)
              (densityLevel sin3 low)
              (densityLevel sout7 low) ...)))
```

Figure 2: Part of an example description of a PDDL initial problem. The goal is to achieve low density in all the streets.

APTC UTC Model

IAS model is a good baseline for our goals, since it is based on a simple traffic model that can be modified by learning. However, IAS actions' descriptions are limited since they only allowed each junction to set one green traffic light at each time step. In order to generate a better model, we would have to study the different cases. The next modeling step takes into account all the possible combinations of density levels and traffic lights that can be set to green at the same time. Finally, it is necessary to guess the effects of applying an action. For instance, in a four-ways junction, if only one incoming street has high density and that traffic light is set to green, we need to guess the density levels of the four outgoing streets after applying this action in order to define the action effects. Clearly, it becomes a hard knowledge engineering task, given the amount of different alternatives, and the variety of behaviors in different junctions and traffic/weather/day conditions.

Also, it is impossible to achieve all goals, i.e., low density in all the streets, under some traffic circumstances. IAS did not return any plan in these cases, which degraded the system's performance. To solve this problem, we transform the hard goals to soft ones following the compilation proposed by Keyder and Geffner [2009]. Individual plans' quality does not necessarily relate to overall quality, since the latter can only be measured at the end of the execution (with metrics such as average waiting time, or pollution). Therefore, we will again use an scheme where we assume that executing more actions implies better performance. Our final domain is composed of ten actions. Only the most basic cases are contemplated in these actions. Then, we propose to automatically update that planning domain by learning the city dynamics, generating junction-based actions that could lead the system to better performance.

APTC Architecture

APTC architecture is based on IAS's and PELEA (Guzmán et al. 2012) architecture and comprises five modules: EXECUTION, MONITORING, PLANNING, MODEL LEARNER and CITY SPLITTER. It is shown in Figure 3.

The EXECUTION component receives an initial AP domain D along with other APTC parameters (M and L). The monitoring rate M indicates how often the city is observed and the duration of the applied actions, i.e., phase sequence. The learning rate L refers to how much time will elapse between two learning episodes. EXECUTION is connected with

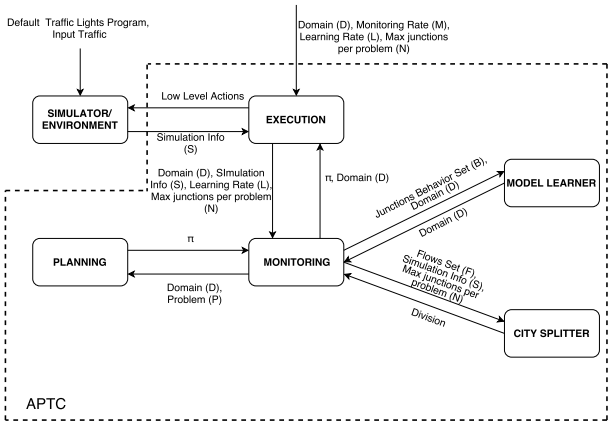

Figure 3: Planning and execution architecture that includes model learning capabilities.

the SIMULATOR that sends the simulation info S every time step and translates the planning actions to low-level actions applied to the traffic lights. S is a directed weighted graph that contains the geometry of the network, $S = (J, E)$. J is the set of vertices (city junctions) and E are the edges that connect those vertices (city street sections). The edges that leave the network to vertices outside it are connected to artificial vertices. An edge $e \in E$ connects two vertices $j_1, j_2 \in J$ if there is a street section that goes from j_1 to j_2. Each $e \in E$ has an associated weight $w(e)$ with values from 0 to 1. At each time step, the weights represent the current density level ratio of that street section. Thus, the structure of the graph has a static component (represented by J and E) and a dynamic component (represented by the weights). At every M steps, EXECUTION asks MONITORING for a plan π and a domain D. If it receives a plan, it is translated from PDDL actions to low-level traffic lights control signals that are sent to the SIMULATOR for their execution. Otherwise, it returns the empty set, so the default traffic light's program in the simulator decides the next actions on the traffic light.

MONITORING is described in Algorithm 1. MONITORING receives a domain, D, the information on the city network, S, the parameter L and the maximum junctions per problem N. It returns a plan π and an updated domain D. It maintains and update two sets based on S: F (Flows) and B (Junction's Behavior), as well as the last executed plan π. F contains information on how the cars traverse the city network and it will be used by the CITY SPLITTER module. B contains information related to the effects of applying an action at a traffic light, and it will be used by the MODEL LEARNER module. The next subsections describe these two modules in detail and how they update D and generate a set of problems $\mathcal{P}$. If a high density street is detected by MONITORING, it automatically generates $\mathcal{P}$ based on the network's state S and the information returned by the CITY SPLITTER. Then, the PLANNER module is called asynchronously with D and every problem $P \in \mathcal{P}$, and the resulting plans are concatenated and returned. We use the notation $\pi_1 \oplus \pi_2$ to represent the concatenation of two plans.

CITY SPLITTER returns a set of disjoint planning tasks in terms of junctions. There is a single traffic light program per junction that controls the incoming streets. The solution of each problem will be composed of actions that affect the junction traffic lights. Since there is only one action per junction and the junctions are only present in one planning problem, we can concatenate the plan of each individual planning task and all these actions are executed in the network in parallel. This strategy will set to green traffic lights that are predicted by the model to have high density levels in the next M (monitoring rate) seconds (starting at the beginning of the planning episodes). The result of this strategy is that green waves are created during these M seconds which can be considered as equivalent to controlling the offsets in classical approaches. The system uses planning to take into account the effects of setting to green a traffic light and how it will affect the surrounding junctions. This is a key difference with respect to reactive systems that usually only consider one traffic light.

Algorithm 1 MONITORING(D, S, L, N)

Inputs: domain D, graph S, learning ratio L, max junctions per problem N
Outputs: π, D

```
B, F, π ←RETRIEVE()
F ←UPDATEF(S)
B ←UPDATEB(S, π)
if simulation-step mod(L) = 0 then
    D ←MODEL LEARNER(B, D)
    A ←CITY SPLITTER(F, S, N)
    F, B ← ∅
if ∃e ∈ E, ISHIGH(w(e))=True then
    P ←GENERATEPROBLEMS(A, S)
    for P ∈ P do
        π_P ←PLANNING(D, P)
    π ← π_1 ⊕ π_2 ⊕ ... ⊕ π_n
else
    π ← ∅
STORE(B, F, π)
return π, D
```

Model Learner

In stochastic environments such as UTC, the world model does not perfectly fit the real world. Also, in the particular case of UTC, most works assume that the model of each action is shared by all network junctions at any time step. But this assumption does not hold in most cases. To overcome these problems we propose to apply learning techniques to continuously adapt and improve the planning model as the actions are executed in the environment. This adaptation starts from the observation of the real effects produced by the execution of each action at each junction in the environment. Every M steps, MONITORING observes the executed action at each junction, the preconditions that hold at that time step, and, at the next MONITORING cycle, the effects after applying that action. Preconditions and effects are related to the density levels of the incoming and outgoing streets of the junction.

Then, the junctions' behavior set B is updated by function UPDATEB. It takes as input the current state of the

network in S and the last executed plan π, and updates the set B. B is composed of tuples b=$\langle j, a, \hat{p}, \hat{o}, f\rangle$ tuples, where $j \in J$ is a network junction; a is an action; $\hat{p}$=$\langle iN, iE, iS, iW, oN, oE, oS, oW\rangle$[2] is the preconditions vector with as many positions as street sections getting in (i) or out (o) of j, with their density values before applying the action; $\hat{o}$ is the effects vector with the same structure as $\hat{p}$, but with the observed density values after applying a; and f is the number of times that the same values for the tuple $\langle j, a, \hat{p}, \hat{o}\rangle$ have been observed. This data is used to compute the most likely effects after applying an action at a particular junction. We will follow the same determinization principles used by Yoon, Fern, and Givan, using only the most frequent effects o for each tuple $\langle j, a, \hat{p}\rangle$ in order to build the new actions. Other determinization schemes could be used in order to apply planning under uncertainty. Table 1 shows an example of a possible set of observations B. Given this data, when `oneHigh` (a) is executed with $\hat{p}$=$\langle h, l, l, l, l, l, l, l\rangle$, $\hat{o}_{a,\hat{p}}$=$\langle l, l, l, l, l, l, l, l\rangle$ are its most frequent effects.

Table 1: Example of B for a specific junction. It shows the frequency with which a set of effects' values $\hat{o}$ are observed after applying an action a with a given vector of preconditions' values $\hat{p}$. h is used for high density and l for low.

j	a	$\hat{p}$	$\hat{o}$	f
$j6$	oneHigh	$\langle h, l, l, l, l, l, l, l\rangle$	$\langle l, l, l, l, l, l, l, l\rangle$	9
$j6$	oneHigh	$\langle h, l, l, l, l, l, l, l\rangle$	$\langle l, l, l, l, l, h, l, l\rangle$	7
$j6$	twoHighIF	$\langle h, l, h, l, l, l, l, l\rangle$	$\langle l, l, l, l, l, l, l, l\rangle$	3
$j6$	twoHighIF	$\langle h, l, h, l, l, l, l, l\rangle$	$\langle l, l, l, l, h, l, l, l\rangle$	7

The MODEL LEARNER module takes as input the previous working domain D and a new set B generating as output a new domain D that includes new actions corresponding to the most frequent observed cases in B. If there is any previous learned action, it is removed from D. So, let us assume that, for each tuple $\langle j, a, \hat{p}\rangle$, $\langle j, a, \hat{p}, \hat{o}\rangle$ is the most frequently observed tuple. a is the action in the original domain D (where st(a) are its static preconditions), $\hat{p}$ are the observed dynamic preconditions, and $\hat{o}$ are the most frequent effects. Then, a new action a' is generated as: a'=$\langle$pre(a'), eff(a')$\rangle$. It does not have parameters, and pre(a')=st(a) $\wedge$ $\hat{p}$ and eff(a')=$\hat{o}$. Since this action is defined for a particular junction, a is maintained in the domain.

The new action is added to the domain model if no conflicts are found. We define a conflict between preconditions $\hat{p}$ of an action a and its most common effects $\hat{o}$ if: a) none of the input streets has high density; or b) when $\hat{p}$=$\hat{o}$. For example, a new action would not be added to the planning domain if $\hat{p}$=$\langle l, l, l, l, h, l, l, h\rangle$, or if $\hat{p}$=$\hat{o}$=$\langle h, l, l, l, l, l, l, l\rangle$.

An example of a learned action from B in Table 1 is shown in Figure 4. The action's preconditions are the conjunction of st(a) and $\hat{p}$. The effects reflect the most frequent density levels after applying a. Note that the action is completely instantiated, given that it refers to a specific junction and a set of specific preconditions. After learning the model, APTC has some new actions that represent the real

[2]This is an example of a junction with four cardinal directions (N, S, E, W).

```
(:action J6-twoHighIF-h-l-h-l-l-l-l-l
 :parameters ()
 :precondition (and (goes-into sin1 j54)
                    (goes-out sout4 j54)
                    (densityLevel sin1 high)
                    (densityLevel sout4 low)
                    ...)
 :effect (and (densityLevel sin1 low)
              (not (densityLevel sin1 high))
              ...))
```

Figure 4: Part of an example description of a learned PDDL action for action twoHighIF in junction J6 from Table 1.

behavior of each junction for a given time period, instead of using a common action that tries to describe the behavior of all junctions in the city at all time steps. Given that the domain model is dynamically learned, it can be adjusted to sudden changes in traffic conditions and automatically return to "normal" conditions when needed. This solves some of the problems that classical approaches face; dynamically adjusting to unexpected situations.

City Splitter

The second contribution of this paper is the use of distributed planning to help scaling up AP when solving UTC problems. The goal is to improve the system's performance and scalability by factoring the whole network into a set of areas $\mathcal{A}$. Each area $s \in \mathcal{A}$ is a subgraph of S such that given two areas s_1=(J_1, E_1), s_2=$(J_2, E_2) \in A$, $J_1 \cap J_2$=$\emptyset$. Therefore, areas have disjoint subsets of junctions. Some edges in the limits of an area can be shared with neighbour areas. These edges are connected to artificial vertices in each area that represent either the vertices outside the network or the junctions in another area. Thus, the same edge can appear in two different areas. Once, a new division in areas $\mathcal{A}$ is computed, APTC generates a set of problems, one per area, that are independent- and asynchronously solved by a planner. The resulting plans can be safely concatenated since they correspond to independent junctions.

The areas could be divided in regular areas following a naïve domain-dependent approach. However, as we show later in the experiments, this leads to a worse performance due to the loss of a property of traffic networks: emergent traffic flows. Since the densities are propagated through the effects of the actions, a bad city partition (split) will not correctly propagate those densities (flows) over different areas. Therefore "green waves" can not be handled.

We overcome this problem by detecting the vehicle flows, taking advantage of the continuous monitoring of the city traffic. APTC joins in a single area those streets and junctions that conform a flow. We say that two street sections with corresponding edges in E, e_1, e_2, are *connected* if there is a pair of junctions $j_1, j_2 \in J$ and: e_1 enters j_1; and e_2 leaves j_1 and enters j_2. If a street e_1 with high density at t_1 is connected with another street e_2 with low density at t_1, and e_2's density becomes high at the next time step t_2, there might exist a flow from junction j_1 to junction j_2. APTC stores these potential transitions in a set of flows F that is updated by function UPDATEF. Each element $fl \in F$ is a tuple fl=$\langle j_1, j_2, f\rangle$, where j_1 is the first junction that vehi-

cles traverse, j_2 the junction cars arrive at and f the number of times that this flow has been observed in the last simulation steps.

F will be used for possibly splitting the city every L steps of simulation. The complete set of tuples in F form a directed graph G=$(J', E') \subseteq S$=(J, E). J'=$\{j|(j, j', f) \in F \vee (j', j, f) \in F\}$; i.e. set of junctions that appear in tuples in F. And E'=$\{(j_1, j_2)|j_1, j_2 \in J, (j_1, j_2, f) \in F\}$ such that $w(e)$=f if e=$(j_1, j_2) \in E'$ and $(j_1, j_2, f) \in F$. All cycles are removed from G to make it acyclic by using the algorithm in (Johnson 1975). In order to maintain the junctions involved in a flow in the same area (and thus planning problem), we look for the maximum-length disjoint paths in G. This can be done in polinomial time, since G is directed and acyclic (Fortune, Hopcroft, and Wyllie 1980). We use the Python package NetworkX[3] for the computation of the maximum length paths as well as the removal of cycles. Algorithm 2 presents the procedure to divide the city.

Algorithm 2 CITY SPLITTER(F, S, N)

Inputs: Set of flows F, graph S, max junctions per problem N
Outputs: $\mathcal{A}$

$U \leftarrow$GETJUNCTIONS(S)
Flows $\leftarrow$MAXLENGTHPATHS(F)
$\mathcal{A} \leftarrow \emptyset$
for each flow **in** Flows **do**
  if `length`(flow)$\geq N$ **then**
    while flow $\neq \emptyset$ **do**
      $\mathcal{A} \leftarrow \mathcal{A} \cup \{\text{flow}[..\mathcal{N}]\}$
      $U \leftarrow U \setminus \{\text{flow}[..N]\}$
      flow $\leftarrow$ flow$[N..]$
  else
    while `length`(flow) $< N$ **do**
      flow $\leftarrow$ flow$\cup$NEIGHBOUR(flow, U)
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{\text{flow}\}, U \leftarrow U \setminus \{\text{flow}\}$
while $U \neq \emptyset$ **do**
  $div \leftarrow \{\text{pop}(U)\}$
  while `length`(div) $< N$ **do**
    $k =$`neighbour`(div, U)
    $div \leftarrow div \cup \{k\}$
    $U \leftarrow U \setminus \{k\}$
  $\mathcal{A} \leftarrow \mathcal{A} \cup div\,, U \leftarrow U \setminus div$
return $\mathcal{A}$

For each flow returned by MAXLENGTHPATHS(F), a partition is generated if it has N elements, the maximum junctions per problem. This parameter affects the planning time and will be discussed in the next section. If the flow includes a higher number of junctions than N, the flow is separated. The first N junctions of the flow are inserted into $\mathcal{A}$ and the remaining junctions ($[N..]$) are the flow that needs to be splitted. If flow has a lower number of junctions than N, the function NEIGHBOUR(flow, U) inserts in flow a junction that is not already part of $\mathcal{A}$, $j \in U$ (the set of unassigned junctions) and it is connected to the last junction in flow. This is done until flow has N junctions. After all junctions in Flows have been assigned to an area and the area added to $\mathcal{A}$, the remaining junctions in U are separated, trying to group neighbour junctions. Finally, the algorithm returns $\mathcal{A}$, that MONITORING will use to generate the different problem files (one per area in $\mathcal{A}$).

[3] `https://networkx.github.io/`

Evaluation

We have used SUMO (Behrisch et al. 2011) for the experiments, an open source traffic simulator. It allows users to define networks, demand and traffic lights control programs. The evaluation is conducted in a grid network similar to the ones present in many cities. The network is composed of 100 junctions and 400 streets. We simulated five hours of a realistic city behavior. This scenario is described in Figure 5. The first 30 minutes of simulation introduces an input flow of cars following a uniform distribution in the city with an average frequency (a vehicle enters the city) of two seconds. With this frequency and distribution, none or just a few congestions are expected to occur. After that, there is a one hour period in which some cars enter the city by two fixed junctions and want to go to the work center. We do this in order to simulate the real behavior of a work day in which most of the people access their job through some main roads. Later, vehicles leave the work and others want to access the stadium in order to attend an sport event. This situation leads to big congestions and the peak of cars in the city is reached at that point. Finally, people leave the stadium and the simulation finishes.

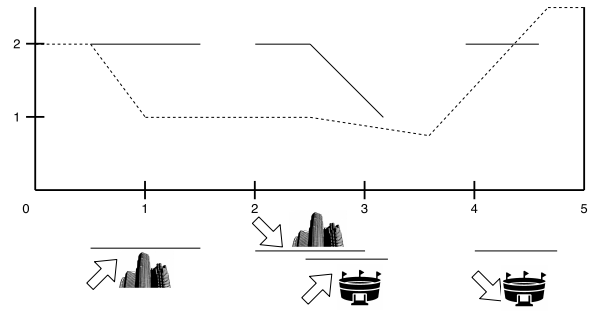

Figure 5: Simulation scenario. The x axis represents the time in hours and the y axis how many cars enter the city every second. The dotted line represents the frequency of cars with random start and destination. The continuous line represents the frequency of cars with fixed start and destination.

The total number of vehicles introduced in the city network is $17,400$ which is way beyond to what other AP-based UTC approaches have reported. For instance, Vallati et al. [2017] used 10 junctions, while our network comprises 100 junctions. Also, Gulič et al. [2016] introduced $5,000$ vehicles in their biggest experiments. Furthermore, most works have used shorter simulation steps than the one reported here.

We compare APTC with three other strategies: STATIC, REACTIVE and IAS (Gulič, Olivares, and Borrajo 2016). STATIC corresponds to the default system used by SUMO and it represents the standard one used in most cities. REACTIVE can be seen as a simplification of SCOOT that does not consider offsets of neighboring traffic lights. IAS does

not have any learning component and only calls the planner when a vehicle has been stopped for a long time. IAS stopped the simulation until a plan was found while APTC does not stop the simulation. We also compare APTC against the starting planning domain without any domain model learning to test whether updating the planning domain represents an improvement or not. We will refer to it as FIXED. Finally, all the AP-based approaches are run twice: one with a MANUAL division and another one with a FLOWS split, the one presented in this paper. We do it in order to test if our factoring criteria works well by itself. We have also tried to compare APTC with the PDDL+ representation employed in (Vallati et al. 2016). However, in our experiments their model is not able to scale up to the networks we use in the time limits we need. Unfortunately, we cannot compare our system against commercial products (e.g. SCOOT).

The values of the parameters used by APTC are the following. Monitoring ratio **M** is the number of steps (seconds) elapsed between two monitoring episodes. This parameter affects the length of the executed actions among others. It has been fixed to 30 seconds, a common cycle in the static traffic lights programs. Learning ratio **L** is the number of steps (seconds) elapsed between two learning episodes. This parameter determines how fast the system is going to react to the changes produced in the environment, updating the planning domain and problems. It has been fixed to 300 seconds (five minutes) in this experiment. We consider that five minutes is a reasonable time to react against changing traffic conditions in the real world. Maximum number of junctions per problem **N** has been fixed to five junctions since it is the maximum number of junctions that a car can traverse in 30 seconds (M) in the experimental city network. This number also allows us to have short planning times.

We use the following metrics: the total amount of CO_2 emitted; the total number of cars that arrive at their destination within the simulation time (DC); the average waiting time of each vehicle (AWT); and the average travel time (ATT). We report the percentage of improvement of each configuration against the base system, STATIC. All the experiments were ran on a Ubuntu machine with Intel Core i7-410U running at 2.00 GHz. All the systems using Automated Planning use Lama 2011 (Richter and Westphal 2010) with a time limit of 20 seconds in order to leave at least 10 seconds to execute the plan. Table 2 shows the results for the simulated scenario. As we can see, APTC Flows, outperforms the rest in all the measured metrics. It is able to reduce the pollution and the waiting and travel times of the vehicles in the city. It also virtually allows all vehicles to reach their destination. It means that the congestion after the end of the sport event has been successfully solved. APTC also outperforms FIXED, the version that does not update the planning domain, showing that learning the city dynamics and how the vehicles traverse the city is in fact a big advantage. Furthermore, the flow-based city split is better than manually dividing the city in equal areas regardless of the planning domain used.

APTC's ability to adapt and react to new traffic conditions is depicted in Figure 6. It shows how the number of new generated domain's actions increases when the traffic dynamics change during the simulation. After learning a traffic behavior, these changes decrease meaning that an effective D has been found for the current scenario.

	$C0_2$	AWT	ATT	DC
STATIC	2972	120	188	16355
REACTIVE	-2%	-15%	-7%	$+1\%$
IASMANUAL	-1%	-13%	-5%	$+1\%$
IASFLOWS	-3%	-16%	-13%	$+2\%$
FIXEDMANUAL	-4%	-21%	-15%	$+3\%$
FIXEDFLOWS	-7%	-25%	-16%	$+4\%$
APTCMANUAL	-11%	-30%	-18%	$+5\%$
APTCFLOWS	-13%	-33%	-20%	$+6\%$

Table 2: Percentage of improvement of each system with respect to the STATIC traffic lights program. AWT and ATT are given in seconds, while $C0_2$ is in kg.

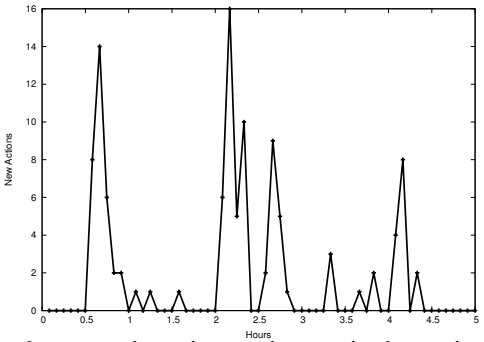

Figure 6: APTC adaptation to changes in the environment.

Conclusions and Future Work

In this paper we have presented APTC, an Automated Planning based system for UTC. As we have shown, there are two main problems when implementing planning systems in this kind of domain. One is related to the generation of accurate planning domains from scratch whose actions reflect the real world dynamics. Our proposal helps designing domain models as well as adapting the models to changes in the environment by updating the model through monitoring and learning. APTC learns the effects of the actions at a junction level and incorporates new actions in the domain. The second problem relates to scalability, where we propose a distributed approach. APTC divides the city according to the detected vehicle flows in order to generate "green waves". Using this approach the system is able to perform well even in large city networks one order of magnitude larger than the ones tested by other AP approaches. The flows-based city splitting criteria and the model learning, the two main contributions of the paper, can improve an AP system even if using them individually. Unifying both techniques we obtain a declarative system for UTC that performs better than other AP-based systems and the static and reactive approaches present in many cities.

In future work, we are interested on analyzing how the different parameters such as the monitoring and learning rates affect the system' performance. Continuous learning

as well as concept drift detection aspects (Gama et al. 2004) could be studied to improve the system' performance and adaptability.

Acknowledgements

This work has been partially supported by MINECO projects TIN2014-55637-C2-1-R and TIN2017-88476-C2-2-R and project PLICOGOR funded by Ministerio de Economía y Competitividad.

References

Behrisch, M.; Bieker, L.; Erdmann, J.; and Krajzewicz, D. 2011. Sumo–simulation of urban mobility. In *The Third International Conference on Advances in System Simulation (SIMUL 2011), Barcelona, Spain*.

Box, S., and Waterson, B. 2012. An automated signalized junction controller that learns strategies from a human expert. *Engineering applications of artificial intelligence* 25(1):107–118.

Bretherton, R.; Wood, K.; and Bowen, G. 1998. SCOOT version 4. In *Proceedings of 9th International Conference on Road Transport Information and Control*.

Cenamor, I.; Chrpa, L.; Jimoh, F.; McCluskey, T. L.; and Vallati, M. 2014. Planning & scheduling applications in urban traffic management.

Fortune, S.; Hopcroft, J.; and Wyllie, J. 1980. The directed subgraph homeomorphism problem. *Theoretical Computer Science* 10(2):111–121.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of AI Research* 20:61–124.

Fox, M., and Long, D. 2006. Modelling mixed discrete-continuous domains for planning. *J. Artif. Intell. Res.(JAIR)* 27:235–297.

Gama, J.; Medas, P.; Castillo, G.; and Rodrigues, P. 2004. Learning with drift detection. In *Brazilian Symposium on Artificial Intelligence*, 286–295. Springer.

García-Martínez, R., and Borrajo, D. 2000. An integrated approach of learning, planning, and execution. *Journal of Intelligent and Robotic Systems* 29(1):47–78.

Gulič, M.; Olivares, R.; and Borrajo, D. 2016. Using automated planning for traffic signals control. *PROMET - Traffic&Transportation* 28(4):383–391.

Guzmán, C.; Alcázar, V.; Prior, D.; Onaindía, E.; Borrajo, D.; Fdez-Olivares, J.; and Quintero, E. 2012. PELEA: a domain-independent architecture for planning, execution and learning. In *Proceedings of ICAPS'12 Scheduling and Planning Applications woRKshop (SPARK)*, 38–45. Atibaia (Brazil): AAAI Press.

Hamilton, A.; Waterson, B.; Cherrett, T.; Robinson, A.; and Snell, I. 2013. The evolution of urban traffic control: changing policy and technology. *Transportation planning and technology* 36(1):24–43.

Huebscher, M. C., and McCann, J. A. 2008. A survey of autonomic computing degrees, models, and applications. *ACM Computing Surveys (CSUR)* 40(3):7.

Jiménez, S.; de la Rosa, T.; Fernández, S.; Fernández, F.; and Borrajo, D. 2012. A review of machine learning for automated planning. *The Knowledge Engineering Review* 27(4):433–467.

Jiménez, S.; Fernández, F.; and Borrajo, D. 2013. Integrating planning, execution and learning to improve plan execution. *Computational Intelligence Journal* 29(1):1–36.

Jin, J., and Ma, X. 2017. A group-based traffic signal control with adaptive learning ability. *Engineering Applications of Artificial Intelligence* 65:282–293.

Johnson, D. B. 1975. Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing* 4(1):77–84.

Keyder, E., and Geffner, H. 2009. Soft goals can be compiled away. *Journal of Artificial Intelligence Research* 36:547–556.

Martínez, D.; Alenya, G.; Torrás, C.; Ribeiro, T.; and Inoue, K. 2016. Learning relational dynamics of stochastic domains for planning. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling*.

McCluskey, T., and Vallati, M. 2017. Embedding automated planning within urban traffic management operations. In *Proceedings of the 27th International COnference on Automated Planning and Scheduling (ICAPS-17)*.

Mourao, K. 2014. Learning probabilistic planning operators from noisy observations. In *Proc. of the Workshop of the UK Planning and Scheduling Special Interest Group*.

Papageorgiou, M.; Ben-Akiva, M.; Bottom, J.; Bovy, P. H.; Hoogendoorn, S.; Hounsell, N. B.; Kotsialos, A.; and McDonald, M. 2007. Its and traffic management. *Handbooks in Operations Research and Management Science* 14:715–774.

Pasula, H. M.; Zettlemoyer, L. S.; and Kaelbling, L. P. 2007. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research* 29:309–352.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39(1):127–177.

Treiber, M., and Kesting, A. 2013. Traffic flow dynamics. *Traffic Flow Dynamics: Data, Models and Simulation, Springer-Verlag Berlin Heidelberg*.

Vallati, M.; Magazzeni, D.; Schutter, B. D.; Chrpa, L.; and McCluskey, T. 2016. Efficient macroscopic urban traffic models for reducing congestion: a pddl+ planning approach. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*.

Xie, X.-F.; Smith, S. F.; and Barlow, G. J. 2012. Schedule-driven coordination for real-time traffic network control. In *ICAPS*.

Yoon, S.; Fern, A.; and Givan, R. 2007. FF-replan: A baseline for probabilistic planning. In *ICAPS*, 352–360.