On Learning Planning Goals for Traffic Control

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

Abstract

Automated planning deals with reasoning processes where a set of goals must be achieved from an initial state using some actions. Most work on planning assumes goals are given. However, in some domains, we could augment the autonomy of reasoning agents by letting them generate their own goals. In those cases, most previous approaches have pre-programmed a set of rules or equivalent goal triggering mechanisms to generate goals under some pre-defined conditions on the state. Instead, we propose in this paper to learn when goals will appear in the next k time steps in order to start the planning process sooner and improve the system behavior. We have applied our approach to a traffic control planning system. Experimental results show how learning goals can anticipate correctly the appearance of congestions, and correctly solve them. The planning system that uses those learned models of goal generation outperforms a planning system that recently won a competition on autonomous behavior.

1 Introduction

Automated Planning (AP) is the AI discipline that generates plans to achieve goals from initial states. A planner receives as input a set of actions (that indicate how to modify the current state), a set of goals to achieve, and an initial state. Many and varied are the techniques explored to this purpose [Ghallab *et al.*, 2004]. In relation to goals, a common assumption is that goals are given as input to the planning system, and there are very few works that focus on reasoning about goals.

Recently, a renewed interest in the study of autonomous agents has emerged in which goals change dynamically and reasoning about goals becomes essential [Vattam *et al.*, 2013]. Vattam *et al.* propose a Goal Reasoning Analysis Framework that identifies three minimum requirements for goal reasoning: goal representation, goal formulation and goal management. Goal representation concerns attributes and characterizations necessary for defining goals and their dynamic properties, such as urgency, utility or cost. Goal formulation provides mechanisms that enable the agents to generate new goals by themselves. And goal management combines the self-generated goals with possible external ones and, depending on the goal characteristics, choose the next goals to address.

In this paper, we focus on domains where the agent is interested in anticipating when new goals should be generated. That could allow the agent to take into account those upcoming goals when reasoning, as was previously done in [Burns *et al.*, 2012]. The task of anticipating when goals will be generated is a difficult one, since there are all kinds of features that influence their appearance. They range from external features to the agent (as human intentions, or particular configurations of the current state) to internal ones (agent motivations). Here, we propose to learn under which conditions new goals will be generated in the near future. We regard it as goal formulation because the agent is able to generate the next goals based on the current state. We apply the learning system to a particular domain, traffic control. It has recently been shown that AP can be successfully used to improve the behavior of city traffic [Gulič *et al.*, 2015; Vallati *et al.*, 2016].

The paper is organized as follows: the next section formally defines AP tasks; the third section details the traffic-control domain; the fourth section describes an architecture that integrates goal reasoning with AP; the fifth section describes the learning system; the sixth section presents the experimental results; and the last section draws conclusions and outlines future work.

2 Planning Tasks

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 literals that must be true in a state to execute the action and a set of effects (eff(a)), literals that are expected to be added (add(a) effects) or removed (del(a) effects) from the state after execution of the action. The definition of each action might also include a cost $c(a)$ (the default cost is one). 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). The planning task 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 goals are true, $G \subseteq s_n$. Plan cost is commonly defined as: $C(\pi) = \sum_{a_i \in \pi} c(a_i)$.

In order to represent planning tasks compactly, the automated planning community uses the standard language PDDL (Planning Domain Description Language) [Fox and Long, 2003]. A planning task Π is automatically generated from the PDDL description of a domain D and a problem P. The domain defines the actions that agents can perform. The problem describes the specific task to be solved at each reasoning step; i.e., the state objects involved, the initial state and the set of goals to achieve.

This planning model assumes the world is deterministic and the agent has full observability, among other assumptions. In most real-world environments, this is not the case. Actions have stochastic outcomes, and agents have partial observability. There have mainly been two ways to handle uncertainty. Either uncertainty is represented explicitly in the planning model and planners reason with those stochastic models [Bonet and Geffner, 2005], or planners reason with deterministic world models and when execution of some actions fails, the agent replans [Yoon *et al.*, 2007]. In this paper, we will use the second alternative given that, from a practical perspective, it is good enough for the domain we are focusing on.

3 Real Problem Domain

Traffic management is becoming a real problem for most big cities. An inefficient traffic control can lead to increase traffic congestion that degrades quality metrics such as average travel time or city pollution. Thus, there have been some proposals that aim at first predicting traffic and then reduce congestions by controlling traffic lights. This paper focuses mostly on the first task, though we provide some results on using the learned models.

In relation to the task of controlling traffic lights, there have been some proposals, some already implemented. They vary from early works on fixed-time static plans that could even generate "green waves" (simple coordination of several traffic lights in order to increase the traffic fluidity), to more dynamic approaches. Recent traffic-responsive control techniques range from centralized approaches, such as SCOOT [Bretherton *et al.*, 1998] to distributed approaches as UTOPIA [Donati *et al.*, 1984]. Most approaches are based on defining a control program using a mathematical framework. A survey on the area can be found in [Hamilton *et al.*, 2013].

We have shown elsewhere that automated planning could be used to compute long-term control plans to improve the traffic system behavior, obtaining a first place in an international competition [Gulič *et al.*, 2015]. The main advantage of using planning is that the domain and problem descriptions are specified in a declarative language. Thus, even traffic engineers can easily include new actions, sensor information or metrics. In those models, the actions provide more time to the green or red phases. States include information on the city static structure, as street sections, crossings, traffic lights, and their relations, as well as dynamic information, as density of traffic or traffic light states. Goals consist of having a low density in street sections with current high density.

Previous work generated those goals once the density was high at a given time step. In this paper, we propose to predict when the density is going to be high in the next steps in order to anticipate problems and improve even further the system's behavior.

4 Architecture

A possible framework where AP can be integrated with the dynamic management of new goals is a planning, execution and monitoring architecture as the one shown in Figure 1. It is an instantiation of the PELEA architecture [Guzmán *et al.*, 2012]. Initially, the *Execution* module captures the current problem state, *state*. This module receives a planning domain and a problem. The initial goal set could be altered by the *Goal Manager* module. The planning problem, *new problem*, includes the current state and the goals selected by the goal manager. Then, the *Monitoring* module calls the *Planning* module to obtain a plan whose actions are sent to the *Execution* module. Once the actions are executed, the *Monitoring* module receives the necessary knowledge (current state, problem and domain) from the *Execution* module to initialize a new planning-execution-monitoring cycle. The monitoring process can also detect unexpected changes in the state that were previously unknown because of the partial observability. This can result in the generation of new goals, as well. This is the task of the *Goal Formulation* module. Then, the *Goal Manager* module should combine these goals with possible external ones (as the ones given directly by users) to generate the new problem. These two modules, *Goal Formulation* and *Goal Manager*, subsume the funcionality of the *Goals&Metrics* module in PELEA.[1] The *Monitoring* module provides the initial state, while goals come from the reasoning process performed by the goal manager given the new formulated goals and the (optional) external ones. The environment can be substituted by a *Simulator* in some domains, as the one we focus on this paper. A similar architecture, MADBOT, differs in the goal formulation module. Their agents generate goals in response to changes in their underlying drives or motivations [Coddington, 2006] instead of learning goal generation models. Other architectures that integrate planning, execution, and goal reasoning derive from the Goal-Driven Autonomy (GDA) conceptual model [Molineaux *et al.*, 2010], inspired in turn by the INTRO architecture [Cox, 2007].

One of the greatest challenges in our architecture is the *Goal Formulation* module. We have to define *when, why* and *how* new goals should be generated [Vattam *et al.*, 2013]. Instead of using a set of predefined rules that automatically generate goals under certain conditions as in most previous work, an alternative is to apply machine learning techniques. The environment can generate examples from observing its behavior given different types of scenarios. The (state,goal) pairs observed during the plans' execution can become the training examples. Then, a learning algorithm can generate a model, such that given any state returns new goals. We are assuming here that the learning process is performed off-line,

[1] In our architecture, for coherence with respect to PELEA, this module returns the new planning problem. Actually, it only changes the goals of the problem.

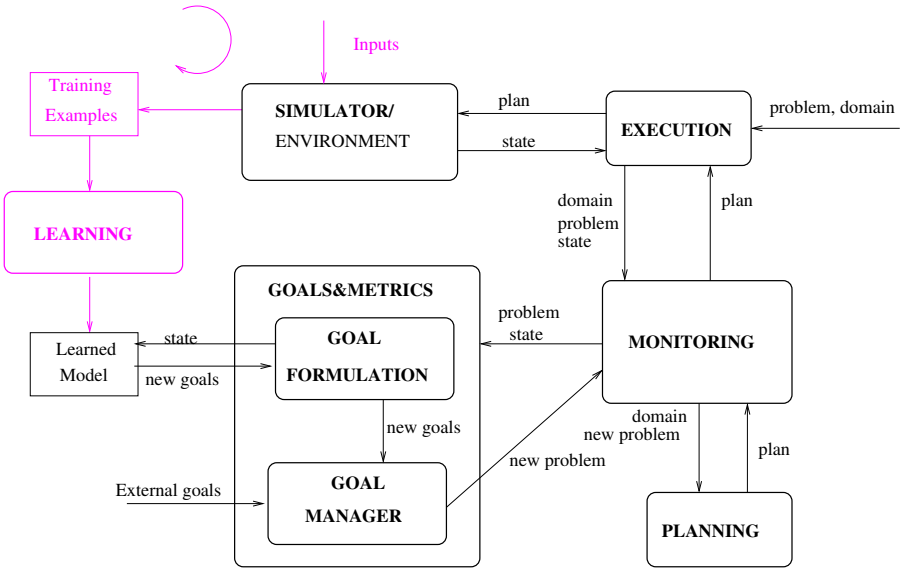

Figure 1: Planning architecture that includes goal formulation and goal learning capabilities.

prior to the actual use of the AP-based system, but it could also be done on-line. The following section details one way to implement such a learning system in our traffic-control domain.

5 Learning new goals

In this work we define the task of learning when goals will arrive; that is, predicting the density level of the streets so we can anticipate their congestion, generating the appropiate goals for the planner. We formulate this problem as a time series prediction one, using Relational Learning in this case. Relational Learning suits AP, because it allows induction over structured examples which can include first-order logical representations, like the ones used in PDDL.

5.1 Representation

The representation is based on a subset of the predicates we use in the planning traffic domain. In order to represent the time steps, we modify some of these predicates, adding the corresponding time steps in the name of the predicates. An alternative representation would be to include time steps and density levels as predicates' arguments. However, complexity of Relational Learning algorithms grows exponentially with the number of arguments, so we tried to reduce the number of arguments. The predicates used for the learning task are shown in Table 1.

We distinguish two types of predicates: the static and the dynamic ones. The static part of the city is represented by the *connection* predicate, that indicates that a vehicle can move from one street section to another. All the *connection* predicates together represent the entire city network. The dynamic part of the city is formed by the state of the traffic lights and the density of the streets. The *openX(tl,st)* predicate represents a green traffic light *tl* located at street *st* at time step X. In our approach, X can take the values from one to three (X previous time steps, or time windows), but it is a parameter that can be modified to extend or reduce the prediction horizon. The *densityLX(st)* predicate indicates that a street *st* has a density level L at time step X. L can take the values *veryhigh, high, moderate, low* and *verylow*. The last predicate of each example, *density(st,l)*, represents the current density level *l* of the street *st*. This will represent the class of each example. Figure 2 shows an example of the information collected at a given time stamp on a street section called A, whose current density is high. We can see that it also had a high density during the last two time steps. Also, another street, B, which is connected to A had a low density in the previous time step 2. The example also has information on traffic lights' states. For instance, `tl1` was open in the previous time step. In summary, each example contains the static and dynamic parts of the entire city network, and its class will be the density level of the street we are trying to predict.

Predicate	Type
density(st,l)	Dynamic
connection(st,st)	Static
open$X$(tl,st)	Dynamic
densityLX(st)	Dynamic

Table 1: Predicates used in the learning task. X represents the time step. L represents the density level.

5.2 Algorithms

For the learning task we are using TILDE [Blockeel and De Raedt, 1998], a system that learns relational decision trees. It receives two files as input: the settings file, where the user can specify the algorithm parameters, as well as defining the predicates and classes; and the knowledge base file, where both the training and test data are included. The output of the learning algorithm is a file containing the resulting relational

```
density(A,high)
connection(A,B)
connection(B,C)
...
open3(tl1,A)
open2(tl2,D)
open1(tl1,A)
...
densityLow3(A)
densityHigh3(D)
densityLow2(B)
densityHigh2(A)
densityHigh1(A)
...
```

Figure 2: Example instance collected on a street section called A.

tree and its translation into rules. It also contains the confusion matrix for the training and test sets. An example output of TILDE is shown in Figure 3, where A represents the example id and the other letters the predicate arguments. A minus symbol predating a variable means that it is new in the tree, while when the variable appears alone, it has to be referenced before. The classes to predict appear in the leaf nodes of the tree between brackets. For example, in the model shown in Figure 3, a high density would be predicted for a street B in two cases: (1) if its density was low two time steps ago, but there exists another street D connected to B whose density was high three time steps ago and was not low in the last time step; and (2) if its density was not low neither two time steps ago nor one time step ago.

```
density(-A,-B,-C)
densityLow2(A,B)?
+-yes: densityHigh3(A,-D)?
       +-yes: connection(A,B,D)?
              +-yes: densityLow1(A,D)?
                     +-yes:[low]
                     +-no:[high]
              +-no:[low]
       +-no:[low]
+-no: densityLow1(A,B)?
      +-yes: [low]
      +-no: [high]
```

Figure 3: Example of TILDE output.

6 Experiments and results

On this work we use SUMO [Behrisch *et al.*, 2011], an open source traffic simulator developed by the German Aerospace Center (DLR). It allows to import or generate not only road networks, but also traffic demand. And it also allows users to define traffic lights control programs. We want to test first, if we are able to build a model to predict the appearance of goals in advance, and then we try to apply the created model to several urban traffic control scenarios.

6.1 Experimental setting

We are using a real city network in our experiments; a grid-like section of Houston downtown, shown in Figure 4. It is composed of 35 junctions, 140 traffic lights and 164 street sections, which presents a high complexity in comparison with many other papers in the field. We have selected five particular street sections to learn from (A to E). We chose these city points due to their different traffic characteristics. C and D are street sections close to the Job Center. B is a point between the Job Center and the main exit of the city. E represents a street section far from the main traffic, while A is a random point with no specific features.

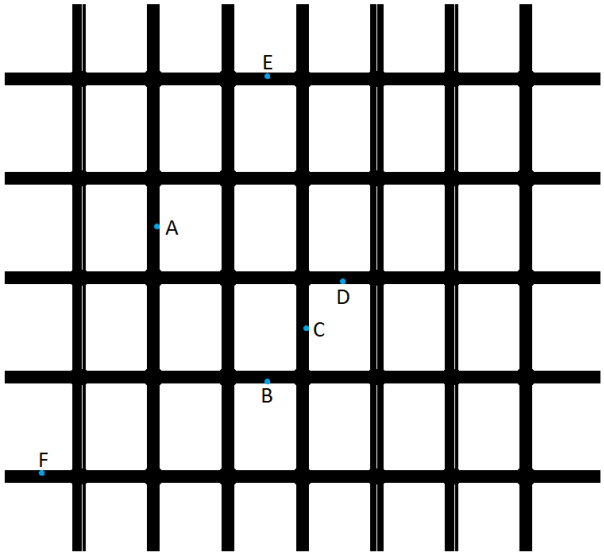

Figure 4: Benchmark network in SUMO. Models are created for points A, B, C, D and E. We assume that a Job Center is located on D. F corresponds to the main exit point of the city.

We have also defined a traffic demand that tries to emulate the real traffic flow of a city for an entire week. So, we define lower vehicle traffic at night, more traffic at rush hours, and higher traffic during week days than in the weekend. A Job Center is included, where most of the cars want to go during the work hours and also a main exit point, to go out of the city at the end of the workday. The rest of the routes are randomly generated. The vehicles may enter the city by any street section and can finish their trip in an inner (parking, mall, office...) or outer point of the network. A summary of the full traffic demand specification is shown in Figure 5.

For the learning task, data is collected every five minutes, which means 2013 instances for the whole week. Five minutes is what we call "time step", the sample frequency. We have chosen this sample frequency as we want to collect traffic data from an entire week, and, at the same time, we want to keep the number of instances low so that TILDE is able to handle them. Each instance stores the static part of the city

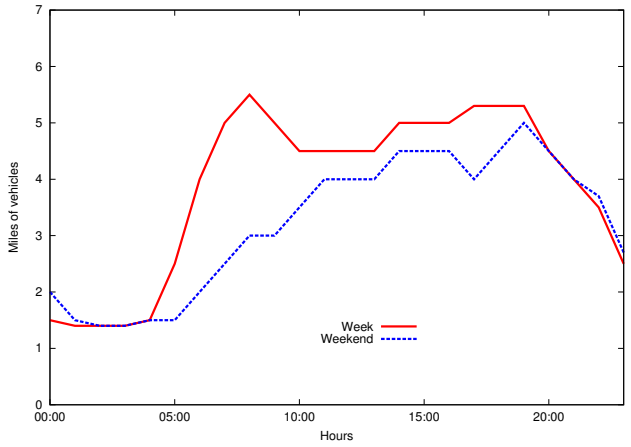

Figure 5: Summary of the generated traffic flows on weekdays and weekends. The y axis represents the number of vehicles that enter the network at each hour, in thousands, and the x axis represents the hours.

previously described, as well as the dynamic component of the state in the last three time steps. We learn one relational model for each street section shown in Figure 4, and then we test with data of the other street sections.

We have also varied the density levels, both in the classes to predict and the predicates used on each instance. We have used two models. One is based on five density levels: *very-high, high, moderate, low* and *verylow*. A second version uses only two: *high* and *low*. All the generated models are pruned, limiting the creation of new branches. For the tree to expand a new branch it is necessary to contain at least 10 instances.

6.2 Results on Learning Goals

In the first experiment, we generated five different models using data from the five selected street sections and the five density levels model. We tested these models in the five street sections to check accuracy and generality of the learned models. The results for this first configuration are shown in Table 2.

	A	B	C	D	E
A	0.90	0.68	0.85	0.77	0.83
B	0.82	0.72	0.79	0.77	0.80
C	0.83	0.66	0.88	0.77	0.81
D	0.80	0.66	0.83	0.85	0.81
E	0.87	0.66	0.85	0.78	0.89

Table 2: Accuracy results using the model obtained with five density levels. Each cell (i, j) represents the estimated accuracy of learning a model with the data extracted at point i in the city and testing that model against the data collected at point j.

We can observe that the accuracy is similar for all the street sections except for B, whose behaviour seems to be more difficult to predict. A and E, the two points away from downtown and the Job Center, present a similar behaviour, as expected.

In the second experiment, we used only two density levels for the class, while we retain the five density levels on the predicates. The results for this second configuration are shown in Table 3.

	A	B	C	D	E
A	0.99	0.84	0.98	0.90	0.95
B	0.94	0.91	0.92	0.87	0.91
C	0.95	0.83	0.98	0.91	0.93
D	0.96	0.83	0.96	0.92	0.94
E	0.95	0.85	0.97	0.91	0.95

Table 3: Accuracy results using two density levels for the class and five on the predicates. Each cell (i, j) represents the estimated accuracy of learning a model with the data extracted at point i in the city and testing that model against the data collected at point j.

The improvement on accuracy is remarkable when we only use two levels of density instead of five in the class. B is still the most difficult point to forecast, followed by D.

In our final experiment, the problem is simplified with only two density levels both for the class and the state predicates. The results for this last configuration are shown in Table 4.

	A	B	C	D	E
A	0.99	0.94	0.99	0.97	0.99
B	0.99	0.95	0.99	0.98	0.99
C	0.96	0.93	0.99	0.97	0.99
D	0.96	0.93	0.99	0.98	0.99
E	0.96	0.93	0.99	0.97	0.99

Table 4: Accuracy results using two density levels for the class and the predicates. Each cell (i, j) represents the estimated accuracy of learning a model with the data extracted at point i in the city and testing that model against the data collected at point j.

We can observe that as we decrease the number of density levels, the complexity of the problem decreases too and the prediction task becomes easier. With only two levels, predicting the density of a street knowing the state of the city in the most recent time steps can be done with a high accuracy, even in street sections which have very different behavior. The final model that will be used in our architecture corresponds with the one generated with the data of point B, which on average performs best. It is shown in Figure 3.

The final experiment focuses on understanding differences in the learning task when we train and test on different days and hours of the day. This data is collected from point B and results of this experiment are shown in Tables 5 and 6. As we can see, the week days have a similar behaviour and they differ from the weekends, as expected. Also, days look alike more than hours.

6.3 Case study

Finally, we want to test whether a traffic control system would improve its performance if it had some predictive model of

	Tuesday	Friday	Saturday
Tuesday	0.99	0.88	0.85
Friday	0.99	0.99	0.94
Saturday	0.92	0.83	0.98

Table 5: Accuracy results of training and testing on different days.

	8:00	20:00	2:00
8:00	0.83	0.75	0.74
20:00	0.66	0.91	0.63
2:00	0.71	0.69	0.96

Table 6: Accuracy results of training and testing on different hours.

the traffic. We will use the same simulation scenarios as before. So, we use the learned model, predicting at every time step the density at each street, using the previous X time steps as input information. If it detects a high density at any subset of street sections, it generates as goals to lower the density of those street sections. These new goals, together with the current state of the traffic, create a PDDL planning problem which is given as input to the planner. Therefore, the system is predicting the appearance of goals in the next X time steps, and the planning process can anticipate to the problems. We will call this new approach `Learning`. We compare it with a `Static` one, in which the traffic lights' programs remain constant. We also compare with the AP approach proposed in [Gulič *et al.*, 2015], co-winner of ARTS-COST competition on *Increasing the resilience of road traffic support systems by the use of autonomics*[2] that did not have any learning component. We will call it `Planning`. This system is the starting point of our approach, so we use the same planning domain and planner, LAMA [Richter and Westphal, 2010].

In the `Learning1` system, the goal prediction is done with the five-minutes-step learned model, and it is triggered (checks for new goals) every five minutes. The `Learning2` system uses the same five-minutes-step model, but it is triggered every fifty seconds (the sample frequency used by the AP approach we compare against). Finally, the `Learning3` system uses a learned model with examples obtained every fifty-seconds, and the learned model triggered also every fifty seconds. Given that TILDE cannot cope with the high number of examples generated during a week sampling every fifty seconds, we randomly selected instances from the different expected behaviours. That is, we randomly selected a subset of instances (the same number as in the previous model) that contains examples of all days and hours of the week from point B, the same street section we used to build the previous model.

The metrics we use to measure the performance of each system are: the number of steps it takes all cars to reach their destinations; the total amount of $C0_2$ emitted by the vehicles; the average waiting time (AWT); the average travel time (ATT); the number of cars on destination at the end (COD), and, if it applies, the number of planner executions (PE).

[2] `https://helios.hud.ac.uk/cost/comp2.php`

For the first experiment, we created a fluent traffic scenario by introducing 5300 cars in 3600 steps. The simulation finishes if all cars reach their destination, or after 5000 steps. The results are shown in Table 7. We can see that there is no substantial difference when the traffic is fluid among the different systems, but our learning approaches outperform the other systems on most metrics. So, when the traffic is fluent, one expects that even the `Static` control program (as in most cities) will perform well. In this traffic situation, the time spent in average per vehicle in a traffic light (AWT) is approximately half of the total time spent in their complete travel (ATT). Given the size of the example network, ATT is around three minutes, while AWT is around a minute and a half.

	Steps	$C0_2$	AWT	ATT	COD	PE
Static	3969	1103	93	172	5300	
Planning	4070	1117	95	175	5300	22
Learning1	3902	1103	92	172	5300	2
Learning2	3881	1090	88	167	5300	15
Learning3	3969	1094	90	168	5300	17

Table 7: Performance of the different control systems with a fluent traffic situation. Steps, AWT and ATT are given in steps (seconds), while $C0_2$ is in kg.

However, in the last experiment, we test the systems on a very congested traffic scenario. It was created by introducing 6000 cars in one hour (3600 steps). The results are reported in Table 8. The columns report the same metrics as the one before.

	Steps	$C0_2$	AWT	ATT	COD	PE
Static	-	2553	582	638	3504	
Planning	-	2187	435	506	4339	48
Learning1	-	2658	609	668	3557	7
Learning2	4070	1265	121	204	6000	46
Learning3	4080	1253	117	200	6000	40

Table 8: Performance of the different control systems with a very congested traffic situation. Steps, AWT and ATT are given in steps (seconds), while $C0_2$ is in kg.

As we can see, even if the `Planning` approach outperforms the `Static` system, only our two last learning approaches can completely solve the traffic congestion. They also improve very substantially with respect to all metrics and all vehicles reach their destination. While, on average, the vehicles spend much more time waiting than travelling in this scenario (relation between ATT and AWT), our system is able to reduce the waiting time to half of the travel time, as in a fluent traffic situation. Thus, it is effectively converting a congested situation into a fluent traffic situation. The reduction of the pollution achieved by learning is quite substantial too: half of the $C0_2$ levels of the other approaches. In fact, they are close to those generated in a fluent traffic scenario. `Learning1` performs worse than `Planning` due to its low number of calls to the planner. Since it only tries to predict future goals every five minutes, and it is a very high

traffic situation, when the streets are already congested, it can be too late for the plans to solve the problem. The difference between the other two learning models is minimum. Therefore, we can conclude that when given a reasonably good prediction model, reducing the checking-for-goals frequency becomes more important than having the same time step for both building the model and checking for goals.

One of the major drawbacks of the `Planning` system is that it calls the planner in a reactive way when a car is stopped for a long time (and then checks the planner actions every 50 steps). The problem of this triggering method is that though SUMO provides this metric, there is no current sensor capable to provide that input in a real life scenario. Our learning systems overcome this limitation, since they only work with density levels, which are easier to provide by current sensor systems.

7 Related work

Most works that apply learning in the context of goal reasoning have focused on the Goal-Driven Autonomy (GDA) conceptual model and its instantiation in ARTUE [Molineaux *et al.*, 2010]. A GDA agent generates a plan to achieve a given goal together with its expectations; i.e., the set of constraints that are predicted to hold in the partial states generated when executing the plan. The agent monitors the environment for discrepancies between its expectations and its observations during execution. If the expectation does not match the state, the GDA agent formulates a new goal using rule-based *principles*, which describe situations where specific goals should be formulated and their relative importance. These rules are hand-crafted by a domain expert, but they can be learned, as well. Powell *et al.* extended ARTUE with the ability to learn goal selection knowledge through interaction with an expert [Powell *et al.*, 2011]. They framed this as a case-based supervised learning task that employs active learning. We also apply supervised learning with a symbolic representation of the learned knowledge. But our learning algorithm is TILDE, a Relational Learning algorithm. Also, our learning algorithm does not have human supervision. The learning concepts also differ; we try to anticipate the generation of new goals based on the current state, while previous works learn the rules for generating goals when the current plan fails.

Jaidee summarized some work on creating GDA agents capable of automatically acquiring knowledge using Case-Base Reasoning (CBR) and Reinforcement Learning (RL) methods [Jaidee, 2013]. In this case, the problem domains are Real-Time Strategy (RTS) games, more specifically DOM and Wargus. Weber *et al.* implemented a method that also uses CBR and intent recognition in order to build GDA agents that learn from demonstration [Weber *et al.*, 2012]. They applied the approach to build an agent for the RTS game StarCraft. Finally, Molineaux and Aha employed a variant of FOIL [Quinlan, 1990] to learn models of unknown exogenous events in partially observable, deterministic environments and show how they can be used by a GDA agent [Molineaux and Aha, 2014]. They implement this learning method in FOOLMETWICE, an extension of ARTUE. Again, our work differs from those in the learning algorithm and the learning task. Closer to our approach is the work presented by [Maynord *et al.*, 2013]. They use TILDE to learn a decision tree for goal prediction in the blocksworld planning domain. In this case, while they learn from world states in isolation, we take into account the time context, as we formulate our problem as a time series prediction one.

In the traffic domain, Relational Learning has been used for other tasks, such as detecting incidents [Dzeroski *et al.*, 1998; Lu *et al.*, 2012]. The task was to find rules that identify incident or incident-free states from collected traffic data in freeways. Instead, we find rules to anticipate high traffic density in urban streets. Lippi *et al.*, like us, addressed the problem of traffic forecasting [Lippi *et al.*, 2010]. However, they proposed a Statistical Relational Learning approach using Markov logic networks instead of TILDE. They manually coded the spatio-temporal dependencies using rules (first-order logic formulae) and the learning algorithm induced the probabilities of those rules. Instead, we learn the rules from scratch.

8 Conclusions and Future work

In this paper we have presented an approach to learn how to predict when new goals will appear. In a goal reasoning context, it can be viewed as goal formulation, one of the current open issues in the community. Using this learning component a planning system can highly increase its autonomy by automatically generating its own goals and planning to anticipate their appearance. We have tested our model in a traffic control domain showing that the ability to anticipate goals can lead to better control performance.

In future work, we would like to integrate the ability to learn how to anticipate goals with externally supplied goals (e.g., by traffic controllers), reactively generated ones (e.g., reactively generating goals), or internally supplied ones (e.g., generated by internal motivations of the system), as well as test the system in larger cities and time periods in order to test its scalability. We would also like to apply the same goal learning approach to other similar domains, as surveillance (satellite), experimental science (rovers), logistics, or transportation (e.g., taxis, packages) domains. We would also like to compare our system with other state of the art methods on traffic control, such as model predictive control (e.g., SCOOT), or other AI-based approaches (e.g., reinforcement learning).

Acknowledgements

This work has been partially supported by MINECO project TIN2014-55637-C2-1-R.

References

[Behrisch *et al.*, 2011] Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. Sumo–simulation of urban mobility. In *The Third International Conference on Advances in System Simulation (SIMUL 2011), Barcelona, Spain*, 2011.

[Blockeel and De Raedt, 1998] Hendrik Blockeel and Luc De Raedt. Top-down induction of first-order logical decision trees. *Artificial intelligence*, 101(1):285–297, 1998.

[Bonet and Geffner, 2005] Blai Bonet and Héctor Geffner. mGPT: A probabilistic planner based on heuristic search. *JAIR*, 24:933–944, 12 2005.

[Bretherton *et al.*, 1998] R. Bretherton, K. Wood, and G.T. Bowen. Scoot version 4. In *Proceedings of 9th International Conference on Road Transport Information and Control*, 1998.

[Burns *et al.*, 2012] Ethan Burns, J. Benton, Wheeler Ruml, Sungwook Yoon, and Minh B. Do. Anticipatory on-line planning. In *Proceedings of ICAPS*, 2012.

[Coddington, 2006] Alexandra M. Coddington. Motivations for MADbot: a motivated and goal directed robot. In *Proceedings of the 25th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2006)*, pages 39–46, 2006.

[Cox, 2007] Michael T. Cox. Perpetual self-aware cognitive agents. *AI Magazine*, 28(1):32–46, 2007.

[Donati *et al.*, 1984] F Donati, Vito Mauro, G Roncolini, and M Vallauri. A hierarchical decentralized traffic light control system. the first realisation "progetto torino". In *Proceedings of the 9th World Congress of the International Federation of Automotive Control*, pages 2853–2858, 1984.

[Dzeroski *et al.*, 1998] S. Dzeroski, N. Jacobs, M. Molina, C. Moure, S. Muggleton, and W. Van Laer. Detecting traffic problems with ILP. In *Proceedings of the 8th International Conference on Inductive Logic Programming*, volume 1446 of *Lecture notes in Artificial Intelligence*, pages 281–290. Springer, 1998.

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

[Ghallab *et al.*, 2004] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning. Theory & Practice*. Morgan Kaufmann, 2004.

[Gulič *et al.*, 2015] Matija Gulič, Ricardo Olivares, and Daniel Borrajo. Using automated planning for traffic signals control. In *Working Notes of ARTS-COST 2nd competition*, 2015.

[Guzmán *et al.*, 2012] César Guzmán, Vidal Alcázar, David Prior, Eva Onaindía, Daniel Borrajo, Juan Fdez-Olivares, and Ezequiel Quintero. PELEA: a domain-independent architecture for planning, execution and learning. In *Proceedings of ICAPS'12 Scheduling and Planning Applications woRKshop (SPARK)*, pages 38–45, Atibaia (Brazil), 2012. AAAI Press.

[Hamilton *et al.*, 2013] Andrew Hamilton, Ben Waterson, Tom Cherrett, Andrew Robinson, and Ian Snell. The evolution of urban traffic control: changing policy and technology. *Transportation planning and technology*, 36(1):24–43, 2013.

[Jaidee, 2013] Ulit Jaidee. Integrated learning for goal-driven autonomy. Master's thesis, Lehigh University, 2013.

[Lippi *et al.*, 2010] Marco Lippi, Matteo Bertini, and Paolo Frasconi. Collective traffic forecasting. In *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2010, Barcelona, Spain, September 20-24, 2010, Proceedings, Part II*, pages 259–273, 2010.

[Lu *et al.*, 2012] Jian Lu, Shuyan Chen, Wei Wang, and Bin Ran. Automatic traffic incident detection based on nfoil. *Expert Systems with Applications*, 39:6547–6556, 2012.

[Maynord *et al.*, 2013] Michael Maynord, Michael T Cox, Matt Paisner, and Don Perlis. Data-driven goal generation for integrated cognitive systems. In *2013 AAAI Fall Symposium Series*, 2013.

[Molineaux and Aha, 2014] Matthew Molineaux and David W. Aha. Learning unknown event models. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 395–401. AAAI Press, 2014.

[Molineaux *et al.*, 2010] Matthew Molineaux, Matthew Klenk, and David W. Aha. Goal-driven autonomy in a navy strategy simulation. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010.

[Powell *et al.*, 2011] Jay Powell, Matthew Molineaux, and David W. Aha. Active and interactive discovery of goal selection knowledge. In *Proceedings of the Twenty-Fourth International Florida Artificial Intelligence Research Society Conference, May 18-20, 2011, Palm Beach, Florida, USA*. AAAI Press, 2011.

[Quinlan, 1990] J. Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, August 1990.

[Richter and Westphal, 2010] Silvia Richter and Matthias Westphal. The lama planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39(1):127–177, 2010.

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

[Vattam *et al.*, 2013] S Vattam, Matthew Klenk, Matthew Molineaux, and David W Aha. Breadth of approaches to goal reasoning: A research survey. In *Goal Reasoning: Papers from the ACS Workshop*, College Park MD, 12/2013 2013.

[Weber *et al.*, 2012] Ben George Weber, Michael Mateas, and Arnav Jhala. Learning from demonstration for goal-driven autonomy. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.* AAAI Press, 2012.

[Yoon *et al.*, 2007] Sungwook Yoon, Alan Fern, and Robert Givan. FF-replan: A baseline for probabilistic planning. In *ICAPS*, pages 352–360, 2007.