Learning-driven Goal Generation

A. Pozanco, S. Fernández and D. 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

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 have a static view of goals; they are given at start of the planning process and they do not change over planning and/or plan execution. However, in many real world domains, agents need to consider dynamic goal management. In this paper, we propose to increase the performance of planning agents by learning when goals will appear in the near future. The learned predictive models allow agents to perform some kind of anticipatory planning, where the planning process considers not only current goals, but also future predicted goals. We also study under which conditions this anticipatory approach outperforms a standard planning approach. Finally, experiments that support our hypothesis are presented.

Keywords: artificial intelligence, automated planning, goal reasoning, anticipatory planning

Introduction

Automated Planning is the AI discipline that generates plans to achieve goals from given initial states. A planner receives as input a set of actions (that describe how the states are modified by their execution), a set of goals to achieve, and an initial state. Many and varied are the techniques explored to this purpose [16]. In order to make it tractable, classical planning has made a set of assumptions. In relation to goals, a common assumption is that goals are given as input to the planning system and they remain static during the execution of plans.

In recent years the interest in creating autonomous agents for numerous real world applications has greatly increased. Applications range from surveillance purposes to control tasks [1,7,12,15,28]. In these cases the closed-world and static-goals assumptions do not hold any more and reasoning about goals and the changes in the environment becomes essential [29,31]. Therefore, new approaches explicitly deal with dynamic goals. A notable example is the concept of Goal-Driven Autonomy (GDA), inspired by Cox's work [9] and detailed in [20]. GDA is a conceptual model that explicitly considers goal reasoning as a key component of the deliberative reasoning process of autonomous agents. It allows us to design and deploy autonomous agents that can explicitly reason about their goals, identifying when they need to be updated or changed through environment monitoring.

The first works on GDA only generated goals following a set of pre-programmed rules that are triggered under some state's conditions [8]. A human must code all the goal-triggering rules before the system's execution. Some recent works on goal reasoning learn goal formulation without human interaction, extending agents autonomy [19]. All these previous works rely on goal reasoning based on the current state of the world. We want to extend the agent's performance by creating a system that is able to generate and handle not only the current existing goals, but also the possible upcoming ones, given the current and recent states of the environment.

From an Automated Planning point of view, previous approaches trigger planning episodes when the current state and/or goals change (most often state changes). We refer to this paradigm as Reactive Planning, since its behaviour is triggered to react to changes in the environment. Our system is based on Anticipatory Planning [6] that takes into account the possible upcoming – but not currently existing – goals along with the current goals when triggering the planning process.

The main contributions of this paper include:

– The design of a learning system that can predict the appearance of new goals in the near future. The learning system is capable of learning goal's appearance off-line and on-line by collecting learning examples from the plans' execution. In an on-line setting it is able to handle concept drift; when the conditions of goal's appearance change dynamically.

AI Communications
ISSN 0921-7126, IOS Press.

- The design of a goal management system that takes into account the learned goal predictive models to generate new goals.
- The integration of the new goal management system within a cognitive architecture that already integrates planning, execution, monitoring and replanning capabilities. We call this new approach Learning-driven Goal Generation Anticipatory Planning (LGG-AP)
- The evaluation of its performance in a typical domain for goal management, comparing LGG-AP against Reactive Planning, that only reasons with the current goals.
- The analysis of the impact of different relevant parameters that influence the anticipation of future goals.

The paper is organized as follows: the next section formally defines Automated Planning tasks; the third section enumerates the domain characteristics where LGG-AP can be applied; the fourth section describes an architecture that integrates goal reasoning with Automated Planning; the fifth section details the learning component that allows the agent to generate future goals based on the current and past states; the sixth section describes the experimental setting, including the simulator we will use for the experiments; the seventh section presents the results; the eighth section presents some related work; and the last sections present the conclusions and outline future work.

Planning tasks

A classical 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 $\text{pre}(a) \subseteq s$ 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 the 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) [14]. 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. The underlying representation formalism used in PDDL is predicate logic. In order to increase efficiency, planners usually transform that high level representation into a more efficient one, such as propositional logic or other equivalent representations.

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. Most previous works focus on the uncertainty about the actions outcomes, but in this paper we are interested in a recently presented paradigm that focuses on reasoning about goal's uncertainty [5,6]. There have mainly been two ways to handle uncertainty. Either uncertainty is represented explicitly in the planning model and planners reason with those models [4], or planners reason with deterministic world models and when execution of some actions fails, the agent replans [34]. In this paper, we will use a deterministic approach and we will replan either when the set of goals changes or the system predicts it may change.

Anticipatory Domains

In many real world domains agents can improve their performance if they can anticipate and reason about the arrival of future goals. Agents can generate [10] some (future) goals and start trying to achieve them sooner. In order to define the anticipatory behavior, we will first provide the following definitions related to goals.

Definition 1. Goal predicate. *A predicate p is a goal predicate in a planning domain D if any of its instantiations (groundings) appear in the goals list of any of its problems P.*

Given any domain D, in theory any predicate can appear instantiated as a goal of any problem. However, in most domains there is a subset of predicates that are the ones that appear instantiated as problems' goals. For instance, consider a Taxi domain in which a fleet of

taxis have to serve a set of customers' pick-up requests. The domain model defines predicates like `at-car`, `at-customer`, `car-empty` and `in`, but most problems will only use `at-customer` in the goals description. Thus, `at-customer` would be a goal predicate for the Taxi domain. We assume that our system knows the set of all goal predicates, $\mathcal{P}$. It can be either or given by the user or automatically computed by looking at a set of problems. That allows us to define the following set.

Definition 2. All-goals set of a set of predicates $\mathcal{P}$, $\mathcal{G}_\mathcal{P}$. *Set of all instantiated goals that can be generated by instantiating predicates in $\mathcal{P}$ with world objects.*

Goals in $\mathcal{G}_\mathcal{P}$ can appear at any time step from the beginning until the end of the agent's execution. The agent knows which goals could be given (or generated by the agent), but it does not know if or when they will arrive. For example, in the Taxi domain, the all-goals set $\mathcal{G}_{\{\texttt{at-customer}\}}$ of an agent's execution would consist of all potential instantiations of the goal predicate `at-customer`. So, it would be `at-customer(A,locX)`, `at-customer(B,locY)`, ...

Definition 3. Known goal. *A goal $g \in \mathcal{G}_\mathcal{P}$ is a known goal if it has appeared (given or generated) at any time step. $\mathcal{K} \subseteq \mathcal{G}_\mathcal{P}$ is the set of known goals.*

Following the previous Taxi domain example, if at a given time step only the goal `at-customer(A,locX)` has appeared, it would be a known goal, conforming $\mathcal{K}$. We now define two other types of goals needed for our work: active goals and predicted goals.

Definition 4. Active goal. *A goal g is active if the agent is trying to achieve it in the next planning episode. $G \subseteq \mathcal{K}$ is the set of active goals.*

Definition 5. Predicted goal. *A goal g is predicted if the agent's inner model foresees its appearance in the near future. At the moment a goal is predicted, it becomes active and known. G_l is the set of predicted goals.*

The agent should generate plans to achieve those goals in G. Some of them are known goals that the agent has not achieved yet. But, new goals $g \in \mathcal{G}_\mathcal{P}, g \notin G$ could be added over time. The reason why a goal is added to G can vary: a user adds g to the current set of goals at a given time step; a procedure in the planning agent decides to generate it based on the current state of the world, as in some GDA approaches [8]; or a model predicts its appearance in the near future, as we deal with in this work.

We will make some assumptions in this paper on the properties that domains should meet in order Anticipatory Planning to be useful.

- For each goal $g \in \mathcal{G}_\mathcal{P}$, the agent does not know whether g will appear or when g will appear.
- There is at least one goal $g \in \mathcal{G}_\mathcal{P}$ that follows an appearance pattern; i.e., its appearance depends on some features related to the observable state. This is needed so that learning to predict new goals makes sense.
- There is full observability on the appearance of goals. At each time step, the agent can observe the set of new goals that have appeared or have been generated by itself, updating the set of active goals G^t to G^{t+1}.
- When a goal is active, it will not disappear from G until it is achieved by the execution of the plan.
- An increasing penalty is paid at each time step for all goals g_i that have already appeared, $g_i \in G \setminus G_l$, and have not been achieved. Although predicted goals G_l are active, since the agent is trying to achieve them, the agent only pays the penalty for those goals in G that have already appeared. The total penalty for each goal, $p(g)$ can be computed following Equation 1.

$$p(g) = \begin{cases} 0 & \text{if } g \notin G \\ k_g \times \delta & \text{if } g \in G \setminus G_l \end{cases} \quad (1)$$

  where k_g is the penalty associated to the goal g and δ is the difference between the arrival time of the goal and the time when it is achieved by the execution of some action. The total penalty of the whole planning-execution cycle, P, is the sum of the penalties paid by each goal that has appeared over the complete planning-execution cycle, $p(g)$.

As we have mentioned, there are many domains where those assumptions hold. For instance, take a company that provides products to some warehouses. Suppose that each warehouse will generate a new goal of having a given product when they do not have stock of that product. If the company is able to predict when warehouses will run out of a product, it can plan to supply the product before the warehouses ask for it, offering a better service and saving time for the warehouse.

Yet another domain is the case of a smart city that wants to control its urban traffic, as we have previously shown in [26]. In that work the goals consisted on de-

creasing the density level of the busy streets through changing the green and red phases of the traffic lights. We showed that if we are able to learn a predictive model that suggests us the appearance of congestions in the near future, we can incorporate these predicted goals into the set of active goals. Then, we can start the planning process sooner, improving the behavior of the system and leading to less waiting time for the cars and pollution levels for the city.

These domains can be seen as goal maintenance problems [30], where some predicates must hold during execution. But, Anticipatory Planning offers some advantages. First, techniques that perform goal maintenance trigger actions to achieve maintenance goals as soon as the goals do not hold [24]. For example, when the warehouse runs out of a product, the agent will immediately try to achieve the goal of having a specific quantity of the product. Other recent works define proactive agents. An agent will take actions not only in response to a maintenance goal not holding, but also in anticipation of the maintenance goal being violated [13]. The agent reasons about the effects of several actions in the near future and will not take any action that violates a maintenance goal. That is, the agent takes into account that all the maintenance goals hold during the planning process. This can improve the rational behavior of agent systems, but these maintenance goals cannot deal with exogenous events, as our Anticipatory Planning approach does. We store information about the environment and build a model that predicts the appearance of goals (or maintenance goals that will be violated) in the future based on these exogenous events.

Other example domains, which can not be seen from a maintenance goal point of view, are surveillance tasks, as those of police, guards, or drones. If we can anticipate where the security breach will appear (where each security breach will generate a new goal to address it), they can arrive at the place earlier and patrol (or execute a set of actions) where the predictive model suggests. In this paper, we will focus on Unmanned Aerial Vehicle (UAV) domains. UAV's usage is growing in recent years due to their low price, increasing set of programming tools, and versatility. The possible uses of UAV's range from military approaches to different surveillance purposes. In this case we propose a domain in which an UAV performs surveillance tasks on an area that has been discretized in a grid.

The UAV has to serve a set of requests coming from a surveillance center. Each goal is a request of performing a given set of tasks (as taking images) in a specific cell of the grid. To service a request, the UAV must move from its current position to the cell of the request. For example, a request can be taking an image or making some kind of measurement. So, the goal would be `(taken-image <cell-id>)` or `(measured <cell-id>)`. Since several observation requests can appear at the same time, the UAV is provided with a planning component in order to find the best plan to cover all the requests (goals) with the minimum penalty. The goals do not disappear until they are achieved and can be achieved at any time. This differs from previous works that assume a finite horizon [5,6].

All these domains share the same assumptions, so we expect similar results in terms of improvement of performance over a system that does not anticipate to their appearance. Since we had good results in the traffic domain, we wanted to analyze here whether they generalize to domains with similar assumptions on goals.

In the following section we describe the architecture that allows the agent to generate and formulate its own goals based on learning, as well as performing anticipatory planning and executing the generated plans.

Architecture

Given that the agent needs to integrate learning, goal management, planning, and execution, we have based our work in a domain-independent architecture that we had developed, PELEA [18]. We have instantiated PELEA in LGG-AP in order to add goals prediction capabilities to the architecture, goal management based on learning goals appearance and anticipatory planning. A high level view of LGG-AP is depicted in Figure 1. First, we will briefly describe how the architecture works. In the next section, we will describe in more detail the main contribution of this paper: the goal generation component based on learning goal prediction models. We will also describe in the following section the simulator we have built in order to test the system's performance.

Initially, the `Execution` module receives a planning task, as a *domain* and *problem* file, including the initial state and goals (initial active goals G). These files are sent to `Monitoring` and `Planning` components, so that the `Planning` component can generate a *plan* for the initial planning task. The plan consists of actions that the `Execution` module will translate into *low level actions* that will be sent to the `Environment`. As an example, the action `move(UAV,cell1,cell2)` will be translated into the different power values to each of its four engines.

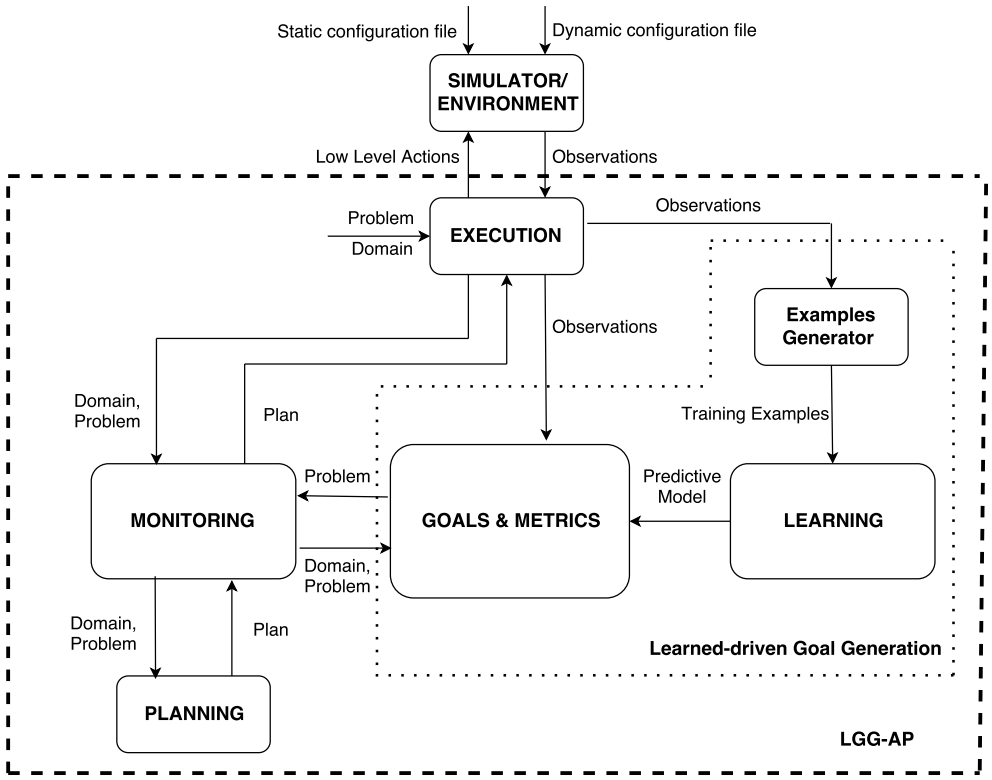

Figure 1. Planning architecture that includes goal formulation and goal learning capabilities.

The `Execution` module periodically receives a set of *observations* from the `Environment`. These observations are translated into a new PDDL problem that is sent to the `Monitoring` component. Execution of actions can be stochastic, but in this paper we will only focus on a specific source of uncertainty; the set of new goals that appears at each time step. If the received state or goals (present in the *problem*) do not match the ones that the `Monitoring` module expected (e.g., a new goal appears and becomes active), this module will call the `Planning` component in order to find a new plan with the state and goals (a new problem). `Execution` also sends the *observations* to the `Examples Generator` and the `Goals & Metrics` components.

At each time step, and before calling the `Planning` module, the `Goals & Metrics` component receives a *problem* from the `Monitoring` component, a set of *observations* from `Execution` and a *predictive model* from the `Learning` component. The `Goals & Metrics` component is in charge of deciding, given a problem, which goals the system should pursue.[1] It can be instantiated in several ways, depending on the goal management approach we want to use. For instance, it can reason about the current subset of goals to plan for in case of oversubscription planning (there is no plan that can achieve all goals with the current resources). We will later describe how it works in our case.

[1]It can also change the metrics to optimize for, though we are not changing metrics here.

Learning-Driven Goal Generation

In this section we describe the Learning-Driven Goal Generation, corresponding to the modules within the dotted line shown in Figure 1. This component allows the architecture to formulate new goals when the system predicts they will appear in the near future. There are three steps in this process: the generation of learning examples; the construction of the predictive model; and the formulation of new goals.

Generation of Examples

Our base idea is to predict future goals based on current and past observations. *Observations* include mea-

surable features about the state and checking whether new goals have appeared. Examples of state features in the case of the UAV can be temperature readings, seismic activity, or any other measurement that our UAV can observe at each cell. Therefore, we will use the state features to generate the attributes of each *training example*. The check about the appearance of a new goal at each cell generates the class of each example. In the case of the UAV, the check at each cell and time step returns true if the surveillance center requested an observation in that cell and time step.

The `Examples Generator` module translates observations into examples in two steps: collecting observations and generating the examples. First, at each time step t during execution, a set of *observations* O_t are collected, one per cell in the grid; $O_t = \{o_{i,t} | i \in [1,m]\}$, where m is the number of cells. Each observation $o_{i,t} \in O_t$ takes the form of:

$$o_{i,t} = \langle \texttt{cell} - \texttt{id}_\texttt{i}, f_{1,i,t}, f_{2,i,t}, \ldots, f_{n,i,t}, g_{i,t} \rangle$$

`cell-id`$_i$ identifies the corresponding cell, $f_{j,i,t} (j \in [1,n])$ represents the value of feature f_j at cell i at time t, n is the number of measurable features, and $g_{i,t}$ will be either true (if the goal has appeared at time t at cell i) or false otherwise.

The learning task consists on building a predictive model for each cell i of whether the system expects a goal to appear at i in the near future. Therefore, each example should contain information about several observations in the past S time steps, as well as the information on whether the goal has appeared in exactly H time steps into the future. S defines the number of previous time steps that will be taken into account when making the prediction, and H refers to the prediction horizon. So, our learning system uses two parameters whose values we will study in the experimental section.

Second, a *training example* can be generated for each cell i and time step t from a set of environment *observations*:

$$\mathcal{E}O_{i,t} = \langle O_{i,t-S}, O_{i,t-S+1}, \ldots O_{i,t-1}, O_{i,t}, O_{i,t+H} \rangle$$

Each $O_{i,k} (k \in [t-S,t]$ or $k = t+H)$ is composed of the feature values observed of the n features at cell i and time k. The union of the values of all observations from $O_{i,t-S}$ to $O_{i,t}$ (corresponding to the measurements from S time steps ago until the current time t) will be the attribute values of each example. And the goal value in $O_{i,t+H}$, $g_{i,t+H}$, will be the class of the example. This is shown in Figure 2. Hence, each training example (at cell i and time step t) can be defined as:

$$\begin{aligned} e_{i,t} = \langle & cell - id_i, f_{1,i,t-S}, \ldots f_{1,i,t}, \\ & f_{2,i,t-S}, \ldots f_{2,i,t}, \ldots \\ & f_{n,i,t-S}, \ldots f_{n,i,t}, \\ & g_{i,t+H} \rangle \end{aligned}$$

The set of *training examples* will be:

$$E = \{e_{i,t} | i \in [1,m], t \in [0,T]\}$$

where T is the number of simulation steps. Therefore, the number of features in each example will be $(S+1) \times n$ (n features, and $S+1$ time steps). And the number of examples will be $m \times T$ (number of cells times the number of simulation steps). The example generation process is independent of the employed representation. In our previous work on the traffic domain, we used a relational (predicate logic) representation where observations were described in terms of a set of literals. Each example was defined as a set of grounded predicates indicating the current and past states of the world and the class of the training example. And we employed a relational learning algorithm that can extract knowledge from these structured examples. In this paper, we use an attribute-value representation instead, where the input data set is typically represented as a single table. Each row in the table is an example and each column is an attribute that represents one particular property of each example.

Building a Goal Predictive Model

Once the system has the examples, it can use a learning algorithm to generate a *predictive model* of future goal appearances. We learn a single model for all cells. In case different cells have different behaviors with respect to goal appearances, the learning system can use the `cell-id` as a feature that can help on making the predictions.

Our approach does not need a particular learning algorithm. The only restrictions are that it has to handle the representation formalism of the examples, and it has to deal with classification tasks (class is discrete). As we mentioned, in our previous work, we used a relational representation. Then, we had to use a relational learning algorithm as TILDE [3], which learns relational decision trees from structured examples based on predicates. In this paper, we use attribute-value representation, so we can use any standard learning technique that solves classification tasks. The input of the learning algorithm is the set of training examples generated in the previous step. The output is a *predictive model*, L. The resulting *predictive model* is provided as input to the `Goals & Metrics` module.

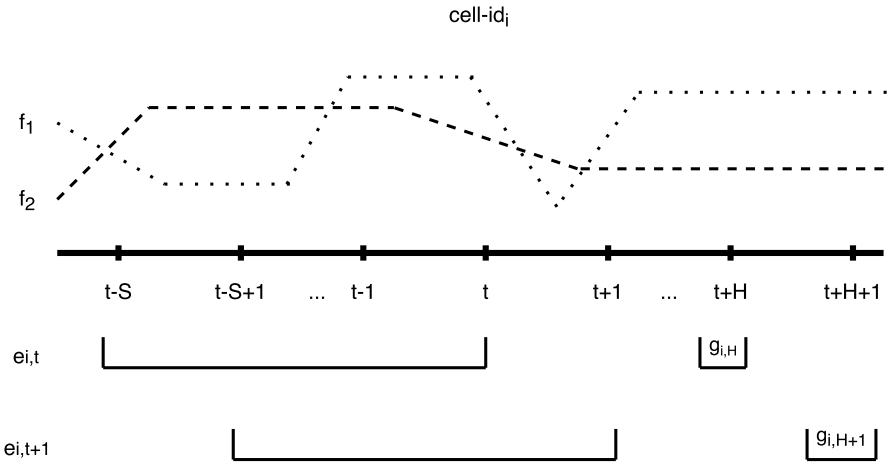

Figure 2. Generation of two training examples $e_{i,t}$ and $e_{i,t+1}$ at a cell i from observations. The attributes of the training examples correspond to the value of two variables f_1 and f_2 in the previous time steps. The class of the training examples indicates whether there is a goal g or not at cell i in the future time step H.

In the case of an off-line learning setting, the system would collect all examples from one or several runs and learn from them. In an on-line setting, as the one in this paper, examples come over time and the system learns every time new examples come. Given that the reasons (pattern) for the appearance of goals can change over time, there might be a concept drift. Therefore, we add new examples and re-train at each time step.

Generation of Predicted Goals

In the last step of learning-driven goal generation, the `Goals & Metrics` module receives as input at each time step t: the current planning task, $\Pi = \{F, A, I, G\}$; a predictive model of goal appearance, L; and a set of environment observations in the previous time steps at every cell in the grid, $\mathcal{E}O_t = \{O_i = \langle o_{1,i}, o_{2,i}, \ldots o_{m,i} \rangle, i \in [t-S, t]\}$. If L predicts future goals G_l (in H steps into the future) based on the observations, the goals in G_l are added to the current set of goals, generating a new set of goals, $G' = G \cup G_l$. The output of this step is an updated planning task that incorporates the new goals, $\Pi' = \{F, A, I, G'\}$.

Experimental Setting

In this section we cover the simulator we have implemented for testing our approach, as well as the input variables and metrics used in the comparison.

UAV Simulator

In our previous work, we used an existing urban traffic simulator, SUMO [2]. Given that we did not have access to a corresponding UAV simulator we built a simulator for our tests. This simulator allows us to define different stochastic scenarios under diverse settings. The simulator receives two files:

- Static configuration file: it contains the static characteristics of the simulation, such as the map size. For the experiments, we have generated a 10×10 grid area like the one shown in Figure 3.
- Dynamic configuration file: it contains the dynamic characteristics of the simulation. In particular, for each time step t, it contains O_t; that is, the set of observations $o_{i,t}$ at time t and cell i. As described above, these observations include the values for each feature f_j at time t and cell i, $f_{j,i,t}$ and the observation of goal appearance, $g_{i,t}$. This scheme allows us to define different scenarios with specific patterns of observations and goal appearances. We consider there could be two kinds of features: those related to the state (e.g. position of the UAV or the images taken); and the ones corresponding to exogenous events of the environment (e.g. the seismic activity and the earth temperature at each cell and time step). The values of these parameters are explained later.

At each time step t, the simulator will receive one action from the `Execution` module. After checking for

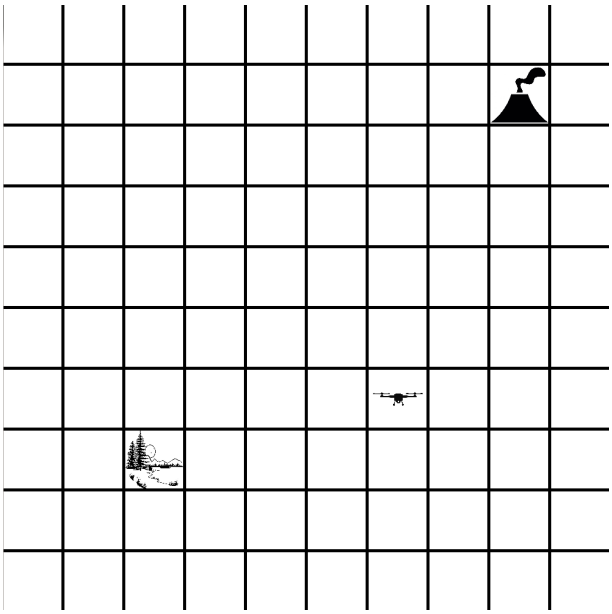

Figure 3. Screenshot from the surveillance UAV simulator. In this example scenario there is an active volcano on the top right side of the grid and a lake on the bottom left.

its validity (it can be applied in the current state), it will change the state of the simulation. Then, it returns to the `Execution` module the observations at time t, O_t. For each cell, these observations include the values of state variables, the values of features related to exogenous events, and the appearance of a goal. The simulator also returns some metrics about the goal achievement process during the simulation. For each goal, it returns its current penalty (described in Equation 1), which we will use later to presents the results.

In the experiments, we have defined two kinds of goal appearance patterns: random and pattern-based. Some goals appear randomly. Other goals follow an appearance pattern that depends on the observations. We will define in each experiment what pattern we are using.

Experimental Variables

First we introduce the parameters we will vary through the simulation and then we present the employed metrics. We will study the impact of varying the main parameters that can affect the LGG-AP performance:

- Anticipation horizon H: we use the values one, three and five.
- Noise in the appearance of the predicted goals: we use the values 0%, 20% and 40%. The noise level represents the probability that a goal does not appear even if the pattern indicates it. We stop at 40%, because with higher values no appearance pattern would be generated.
- Goal ratio: ratio between the number of goals that follow an appearance pattern and the total number of goals. The latter is the sum of the randomly generated ones and the ones that follow a pattern. Both numbers are computed over the whole simulation. We test five different values: $0.7, 0.5, 0.3, 0.2$ and 0.1, corresponding to a high percentage of pattern-based goals down to a low percentage. We fixed the number of goals that follow an appearance pattern to 100. When the ratio decreases, there are more randomly generated goals in the grid.
- Number of goal appearance patterns that occur at the same time step (one per cell maximum): from one to five.
- Number of agents: only one UAV pursuing the goals.
- Exogenous events produced by the simulator: seismic activity and earth temperature. These parameters take values from zero to three. A volcano eruption is correlated with high values of both parameters, i.e., it is likely to occur. The frequency of an eruption will depend on these values and will

be varied over the experiments to test the system's capabilities.

We run each simulation for T=2000 time steps. Since some goals will be generated randomly (subset of $\mathcal{G}_{\mathcal{P}}$ that does not follow an appearance pattern), we run each simulation 10 times to obtain an average penalty. We compare LGG-AP, that updates the set of active goals with the learned goals $G^{t+1} = G^t \cup G_l$, against a Reactive Planning approach that only pursues the current active goals G (not performing any update in the set of goals due to any reasoning process on future goals).

In the experiments the agent pays an increasing penalty of one for each time step when a goal that has already appeared (is active) has not been achieved. The penalty values P obtained by each approach at the end of the simulation depend on the number of goals. Thus, analysis of results can be more difficult to perform. We propose a metric that normalizes the penalty paid by each approach. We denote this metric with $\mathbb{P}$. It is computed as:

$$\mathbb{P} = \frac{(P_r - P_{lgg-ap})}{AG}$$

where P_r is the penalty paid by the Reactive Planning approach, P_{lgg-ap} is the penalty paid by LGG-AP, and AG is the number of all goals that have appeared during the simulation. The resulting number $\mathbb{P}$ can be seen as the saved penalty for each goal $g \in AG$ by using LGG-AP compared to the penalty paid by the Reactive Planning approach. All the experiments were run on a Ubuntu machine with Intel Core i7-4510U running at 2.00 GHz. The results are presented in the next section.

Experiments and Results

In the first experiment we modify three of the parameters that can affect LGG-AP: anticipation horizon, noise in the appearance of predicted goals and the goals ratio. We compare the performance of LGG-AP against the Reactive Planning approach. In the second experiment we study the performance of LGG-AP if we introduce more than one goal appearance pattern. The third experiment focuses on analyzing the concept drift capabilities of our system, testing the agent's adaptation to new conditions, on-line learning and exploiting a new predictive model.

Influence of Parameter Settings in LGG-AP

We begin with a one step anticipation pattern ($H = 1$) with 0% of noise. Figure 4 shows a sample of the observations generated in the volcano cell. In the remaining cells, we provided a random pattern of observations. A pattern-based goal is only generated in the volcano cell, when the value of the two environment features (seismic activity and temperature) is three. In this case, whenever a goal is generated in this cell, the value of both variables in the previous time step was two.

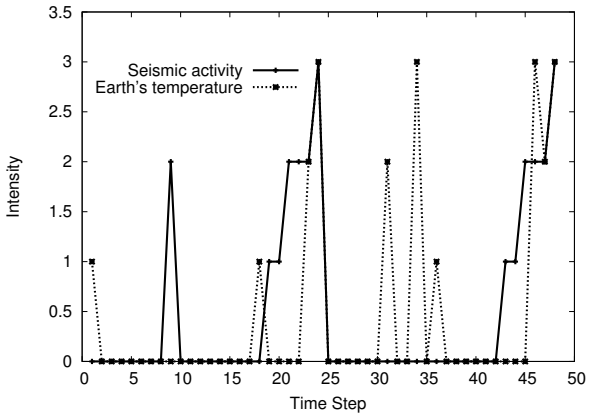

Figure 4. Sample of observations generated in the volcano cell.

Firstly, we want to test the performance of the two approaches over time, and the results are shown in Figure 5. When LGG-AP collects enough data to build a model, it is able to exploit it correctly by anticipating where the goal will be generated and moving the UAV to the potential future goal location (volcano cell). Thus, it outperforms the Reactive Planning approach. The agent's performance is the same using both approaches until LGG-AP is able to build an accurate predictive model around the time step 350. From then on, LGG-AP outperforms Reactive Planning in all the simulation.

We then conduct several experiments where we modify the parameters described in the experimental setting. The results are shown in Figure 6. The difference between LGG-AP and Reactive Planning is bigger when the predicted model is more accurate, as expected. With 0% and 20% noise levels, $\mathbb{P}$ increases as the goal ratio decreases (there are more random goals), until a point where there are many goals where most of them are random. At this point LGG-AP performance becomes similar to that of Reactive Planning, given that both approaches have more opportunities (goals) to decrease

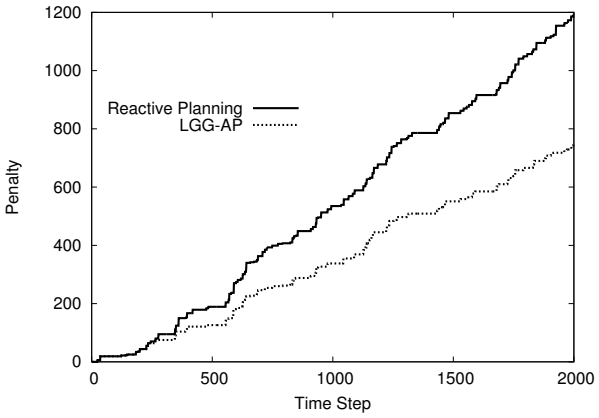

Figure 5. Accumulated penalty of LGG-AP and Reactive Planning. The penalty at each time step of goals is $k_g = 1$, $H = 1$ and noise is 0%.

the penalty paid. Anticipating provides a smaller advantage.

As the noise levels are bigger, $\mathbb{P}$ reaches its best value with less goals and more percentage of them related with the goal appearance pattern. If the goals were introduced randomly without following any appearance pattern, the learning component would not be able to extract any model that correctly predicts goals in the future. In these cases the goal's set of LGG-AP and Reactive Planning would be the same since the set of future goals would be empty. Consequently, their performance would be similar, taking into account only the current goals.

We obtain better results anticipating the goals one or three time steps rather than five. In this case $\mathbb{P}$ values are higher when there are fewer goals. The best anticipation value is closely related to the size of the grid. The experiments show that for this particular 10×10 grid, going for the goals five time steps before their appearance is not as good as doing it three or one time steps before.

Even in the worst case, in which the agent learns an inaccurate predictive model, LGG-AP performance is at least as good as Reactive Planning, independently of the number of goals to achieve or the number of steps that goals are anticipated. This difference in the value of $\mathbb{P}$ between Reactive and LGG-AP is very relevant, since it denotes the difference in terms of penalty per goal, assuming that all goals have an homogeneous associated penalty equal to one. This is not the case in most domains, even in the one that we are presenting here. The real penalty related to a volcano eruption is not the same as the one for not taking a picture of a crop zone. In these cases, the benefit of using LGG-AP instead of Reactive Planning would increase considerably. As an example, in case we had an homogeneous penalty of 1000 for each goal, and 50 goals, a difference of three (as in Figure 6) would mean a penalty of $1000 \times 50 \times 3 = 150000$.

Analysis of the Number of Patterns

In this experiment we study how the number of goal appearance patterns influence LGG-AP. We introduce from one to five volcanoes uniformly distributed over the grid. We compare the $\mathbb{P}$ value fixing the previous parameters to: 0% noise level, 0.5 goal ratio and $H = 3$. The results are shown in Figure 7, which follows the same peak behavior as the previous ones. The system is able to capture every appearance pattern, flying to the places where goals will appear in the future. LGG-AP always outperforms Reactive Planning. $\mathbb{P}$ raises until it reaches the best value when there are three patterns in the grid (three volcanoes). When there are many patterns, we can observe the same behavior as when there are many goals and LGG-AP performance decreases. But the difference with the Reactive Planning approach is still outstanding in any case.

Ability to Handle Concept Drift

In the previous experiments we have shown that LGG-AP outperforms Reactive Planning in terms of the penalty they pay. In those cases, the agent learns patterns that do not change over time and exploits them. In most real world domains these goal appearance patterns change and the agent should discover these new patterns, adapting to them. This is the concept drift paradigm [33].

To test the capability of adapting to new goal patterns, we generate a one time step anticipation pattern in the volcano cell and we change it to a random pattern at time step 850. At that time step, pattern-based goals will not appear any more in the volcano cell and pattern-based goals will appear in the lake's cell. The appearance of goals in the lake will be correlated only with the temperature variable unlike the previous pattern that also depends on the seismic value.

As we can see in Figure 8, LGG-AP starts outperforming Reactive Planning from the beginning of the execution. Around time step 850, when the pattern changes, LGG-AP deteriorates its performance for a period of time. At this point the agent is building two predictive models in parallel: the one that suggests the appearance

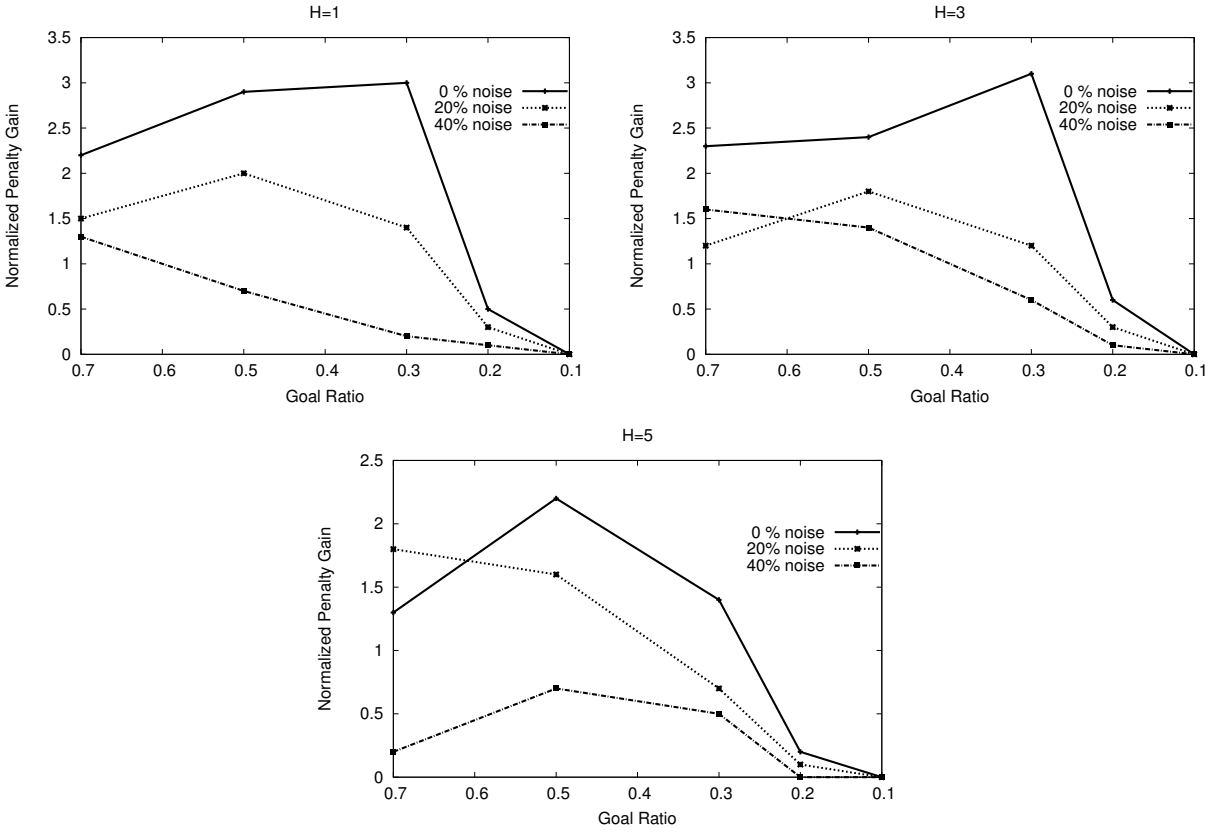

Figure 6. From left to right, top to bottom: one, three and five time steps anticipation. The x-axis represents the ratio between the number of goals that follow an appearance pattern and the total number of goals. The y-axis represents the value of $\mathbb{P}$ for the different noise levels in the prediction.

of goals in the volcano area, which is starting to decrease its accuracy, and the new one that only predicts goals in the lake. These two patterns conflict for a while, leading to similar performance between Anticipatory and Reactive planning. This occurs until the learning algorithm discards the old pattern due to its inaccuracy and returns a new predictive model. After time step 1200, LGG-AP is able to recover and again outperforms Reactive Planning. There is no single moment during the simulation in which Reactive Planning works better than LGG-AP.

Related work

Most works in the context of goal reasoning have focused on the Goal-Driven Autonomy (GDA) conceptual model [9,20]. 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 expectations do not match the observed states or if the current plan fails, the GDA agent can formulate a new goal [11,23]. The first works on GDA formulated new goals using rule-based *principles*, which describe situations where specific goals should be generated [8]. These rules were hand-crafted by a domain expert.

Few works have studied the addition of learning capabilities to the agents in the goal reasoning process. Powell *et al.* extended the ARTUE agent [20] with the ability to learn goal selection knowledge through interaction with an expert [25]. They framed this as a case-based supervised learning task that employs active learning. Unlike their approach, we do not need the interaction with a human to formulate goals, since we learn this ability by collecting examples from the agent execution in an environment, that can be real or simulated.

Without the need of human interaction, Jaidee summarized some work on creating GDA agents capable

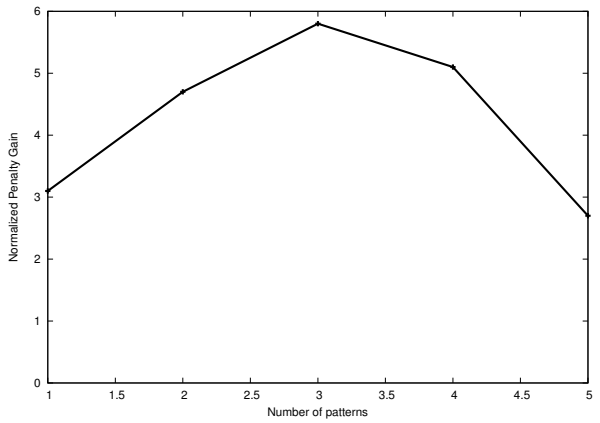

Figure 7. Normalized penalty obtained by using different number of goal appearance patterns. The x-axis represents the number of appearance patterns that could occur at the same time. The y-axis represents the value of $\mathbb{P}$.

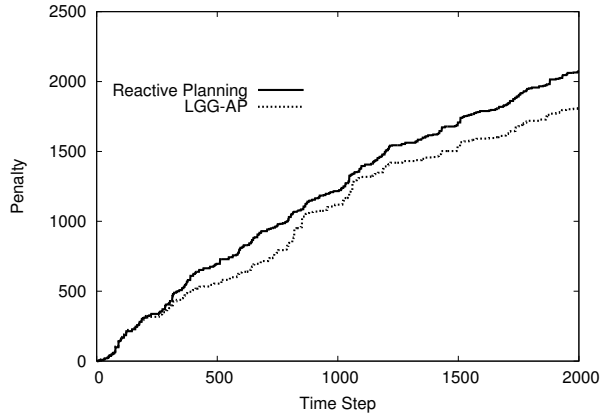

Figure 8. Accumulated penalty paid by Reactive Planning and LGG-AP. The goal appearance pattern changes at time step 850.

of automatically acquiring knowledge using Case-Base Reasoning (CBR) and Reinforcement Learning (RL) methods [19]. 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 [32]. They applied the approach to build an agent for the RTS game StarCraft. Molineaux and Aha employed a variant of FOIL [27] to learn models of unknown exogenous events in partially observable, deterministic environments and showed how they can be used by a GDA agent [22]. They implemented this learning method in FOOLMETWICE, an extension of ARTUE. Maynord *et al.* employ TILDE [3], a relational learning algorithm, to learn a decision tree for goal prediction in the blocksworld planning domain [21]. Finally, Gopalakrishnan *et al.* learn goals from planning traces in planning domains [17]. Our work differs from those in the sense that they are not generating and reasoning with possible upcoming goals, as we do. While they learn from world states in isolation, we take into account the time context, as we are planning with goals predicted by an on-line learning model.

Regarding the concept of Anticipatory Planning we are addressing in this paper, while the idea of using Automated Planning taking into account possible upcoming goals comes from previously presented works [5,6], our work differs from theirs in some aspects. While they assume they know a priori the goal arrival distribution, we are learning it through the collection of examples from the system's execution. Another difference is that they use a special planner that reasons internally with the upcoming goals distribution and its penalties. We propose to use a classical planner, incorporating either the current or the possible upcoming goals and replanning when a new goal appears.

Discussion

In this paper we have presented an architecture that allows the design and implementation of autonomous agents with learning capabilities. Using this architecture for a small UAV domain, we have shown that an agent can discover opportunities and adapt its behavior as the surrounding environment changes following a concept drift approach.

We have gone further in the goal reasoning concept, letting the agent not only reason with the current state of the world but also with the possible near future. If the agent discovers a goal before it really appears, it can start the planning process sooner, improving its performance.

Finally, we have enumerated the requirements that a domain must fulfill in order to successfully apply Anticipatory Planning. We have presented a list of such domains and selected one of them to perform the experiments. Through a surveillance UAV domain, we have carried out some experiments in order to discuss the main characteristics and parameters that affect LGG-AP. The results show that LGG-AP works, in the worst case, as well as Reactive Planning, outperforming the reactive approach in the rest of scenarios. We obtain the best results when the agent has time to reason about the

future goals instead of just be acting all the time. This happens when there are many goals in the grid whether they come from several goal appearance patterns or they are randomly generated.

Future work

In future work, we would like to handle goals that must be achieved on a given time because they disappear, or goals with non homogeneous penalty distribution like in some real world domains. We would also like to explore LGG-AP in the context of multiple agents, instead of only one as we propose in this paper. These agents could have a cooperative or a competitive relation, leading to several goal reasoning challenges and problems.

Acknowledgements

This work has been partially supported by Spanish MINECO funded project TIN2014-55637-C2-1-R. We would also like to thank the anonymous reviewers for their useful comments that helped us greatly improve the paper.

References

[1] M. Ai-Chang, J. Bresina, L. Charest, A. Chase, J.-J. Hsu, A. Jonsson, B. Kanefsky, P. Morris, K. Rajan, J. Yglesias, B. Chafin, W. Dias, and P. Maldague. MAPGEN: Mixed-initiative planning and scheduling for the Mars Exploration Rover mission. *IEEE Intelligent Systems*, 19(1):8–12, feb 2004.

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

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

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

[5] D. Borrajo, R. Fuentetaja, and T. de la Rosa. Anticipatory search as partial satisfaction planning with state dependent costs. In *Proceedings of 4th Workshop on Goal Reasoning (IJCAI'16)*, 2016.

[6] E. Burns, J. Benton, W. Ruml, S. Yoon, and M. B. Do. Anticipatory on-line planning. In *Proceedings of ICAPS*, 2012.

[7] I. Cenamor, S. Nez, T. de la Rosa, and D. Borrajo. Planning for tourism routes using social networks. *Expert Systems with Applications*, 69:1–9, 2017.

[8] A. 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.

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

[10] M. T. Cox. Goal-driven autonomy and question-based problem recognition. In *Second Annual Conference on Advances in Cognitive Systems 2013, Poster Collection*, pages 29–45, 2013.

[11] M. T. Cox, Z. Alavi, D. Dannenhauer, V. Eyorokon, H. Munoz-Avila, and D. Perlis. Midca: A metacognitive, integrated dual-cycle architecture for self-regulated autonomy. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[12] M. de la Asunción, L. A. Castillo, J. Fernández-Olivares, Ó. García-Pérez, A. G. Muñoz, and F. Palao. SIADEX: an interactive knowledge-based planner for decision support in forest fire fighting. *AI Commununications*, 18(4):257–268, 2005.

[13] S. Duff, J. Harland, and J. Thangarajah. On proactivity and maintenance goals. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1033–1040. ACM, 2006.

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

[15] J. García, J. E. Florez, A. Torralba, D. Borrajo, C. Linares-López, A. García-Olaya, and J. Sáenz. Combining linear programming and automated planning to solve intermodal transportation problems. *European Journal of Operations Research*, 227(1):216–226, 2013.

[16] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning. Theory & Practice*. Morgan Kaufmann, 2004.

[17] S. Gopalakrishnan, H. Muñoz Avila, and U. Kuter. Word2htn: Learning task hierarchies using statistical semantics and goal reasoning. In *Proceedings of 4th Workshop on Goal Reasoning (IJCAI'16)*, 2016.

[18] C. Guzmán, V. Alcázar, D. Prior, E. Onaindía, D. Borrajo, J. Fdez-Olivares, and E. 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.

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

[20] M. Klenk, M. Molineaux, and D. W. Aha. Goal-driven autonomy for responding to unexpected events in strategy simulations. *Computational Intelligence*, 29(2):187–206, 2013.

[21] M. Maynord, M. T. Cox, M. Paisner, and D. Perlis. Data-driven goal generation for integrated cognitive systems. In *2013 AAAI Fall Symposium Series*, 2013.

[22] M. Molineaux and D. 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.

[23] M. Paisner, M. T. Cox, M. Maynord, and D. Perlis. Goal-driven autonomy for cognitive systems. In *CogSci*, 2014.

[24] A. Pokahr, W. Braubach, and W. Lamersdorf. Implementing a BDI-infrastructure for JADE agents. 2003. *Exp in Search of Innovation*, 3(3), 2003.

[25] J. Powell, M. Molineaux, and D. 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.

[26] A. Pozanco, S. Fernández, and D. Borrajo. On learning planning goals for traffic control. In *Proceedings of 4th Workshop on Goal Reasoning (IJCAI'16)*, 2016.

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

[28] W. Ruml, M. B. Do, R. Zhou, and M. P. J. Fromherz. On-line planning and scheduling: An application to controlling modular printers. *Journal of Artificial Intelligence Research*, 40:415–468, 2011.

[29] K. Talamadupula, J. Benton, S. Kambhampati, P. Schermerhorn, and M. Scheutz. Planning for human-robot teaming in open worlds. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 1(2):14, 2010.

[30] M. B. Van Riemsdijk, M. Dastani, and M. Winikoff. Goals in agent systems: A unifying framework. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 713–720. 2008.

[31] S. Vattam, M. Klenk, M. Molineaux, and D. 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.

[32] B. G. Weber, M. Mateas, and A. 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.

[33] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23(1):69–101, 1996.

[34] S. Yoon, A. Fern, and R. Givan. FF-replan: A baseline for probabilistic planning. In *17th International Conference on Automated Planning and Scheduling (ICAPS-07)*, pages 352–360, 2007.