Planning for Agents with Changing Goals

Kartik Talamadupula[†] and **Paul Schermerhorn**[§] and **J. Benton**[†] and **Subbarao Kambhampati**[†] and **Matthias Scheutz**[§]

[†]**Department of Computer Science**
Arizona State University
Tempe, AZ 85287 USA
{krt,rao, j.benton} @ asu.edu

[§]**Cognitive Science Program**
Indiana University
Bloomington, IN 47406 USA
{pscherme,mscheutz} @ indiana.edu

Abstract

One of the most important applications of planning technology is guiding robotic agents in an autonomous fashion through complex problem scenarios. Increasingly, real-world scenarios are evolving in ways that require intensive interaction between human actors and the robotic agent, mediated by a planning system. We propose to demonstrate an integrated system in one such problem that falls under the aegis of an urban search and rescue (USAR) scenario. We show a simulation of a run through one such problem where a mobile robot is given certain goals to achieve in a layout of interest, and discuss how the various capabilities of the planner are instrumental in achieving the agent's goals.

Introduction

One of the earliest motivations of Artificial Intelligence was to provide autonomous control to robotic agents that carry out useful service tasks. Application scenarios for these kinds of tasks span a wide spectrum that includes military drones and mules, household assistance agents and search and rescue robots. The level of autonomy desired of such robotic agents can be achieved only by integrating them with planning systems that can plan not only for the initial goals, but also updates to these goals and the state of the world.

Recent years have seen the emergence of fast planning algorithms and systems that can be used to model a large number of the features that distinguish real world applications from theoretical problem scenarios. Key among these features are time, cost, resources, uncertainty and execution failure. Though a few planners have modeled a subset of these features in the past, the scale-up required to support real world timeframes has only come about recently due to the extensive use of heuristic search methods for plan synthesis. Current planners still impose a number of restrictive assumptions in order to support this scale-up; classical planners are the best example of this. The challenge is to identify the essential features when considering planning support for such real-world scenarios.

In this demonstration, we show one such planning system aiding a robotic agent as it navigates a search and report scenario. We describe the environment that the robot is executing in, as well as its goals and actions, in detail. We then provide a brief overview of the planning system that is integrated with the architecture controlling the robot, and detail some extensions that we had to provide in order to enable successful modeling of the application scenario.

Scenario: Search and Rescue

One of the primary applications of robotic agents is in scenarios where a human actor has a plethora of knowledge about the problem at hand, yet cannot act in the world due to inherent dangers to human life – emergency response and firefighting are among the best examples of such scenarios. In such cases, having a robotic agent as part of the team greatly increases the chances of achieving the desired end-goals (rescuing people, putting out a fire etc.) without exposing the human team-member to risks.

In this demonstration, we consider a specific scenario that we had to provide planning support for to illustrate the challenges that crop up when planning for robots in real-world scenarios. This is the urban search and rescue (USAR) scenario – a team consisting of a human and an autonomous robot is tasked with finding and reporting the location of critical assets (e.g. injured humans) in an urban setting (usually a building). A given USAR task may consist of multiple problems, each with different challenges. The human member of the team usually has intimate knowledge of the setting of the scenario, but cannot perform the required tasks due to inherent dangers like fires, gas leaks, collapsed structures etc. The robot is considered autonomous because the human only communicates with it sparingly, to specify the goals that it must achieve and any information that might be deemed useful for its execution in the world. Examples of tasks in the USAR scenario include transporting essential materials to a specified location or entity; and reconnaissance tasks like reporting the locations of trapped or injured humans to the commander and taking pictures of objects or areas of interest. In the following, we present the specific USAR task that we tested our system on in order to illustrate the inherent planning challenges.

Task: Search and Report

In this problem, the robot's main goal is to deliver essential medical supplies to a specific location within the area of interest – during its run, the robot may be given additional information and goals about other assets. The human team-member (the commander) has intimate knowledge of the building's layout, but is removed from the scene and

can only interact with the robot via on-board wireless communication. The robot begins in a long hallway that has doors leading off into rooms on either side. Initially, the robot is unaware that that these rooms may contain injured or trapped humans, and its goal is to reach the end of the hallway to deliver the supplies by a given deadline.

As the robot executes a plan to achieve that goal, the human commander notes that it is passing the rooms and recalls that injured humans may be trapped in these rooms. The commander then passes on this information linking rooms to injured humans to the robot, and specifies a new goal on reporting the location of as many such humans as possible given the time and resource constraints imposed by the achievement of its original goal. In addition, the human commander remembers that rooms have doors, and that these doors must be pushed open in order for the robot to gain access to the room behind that door. The robotic agent (and hence the planner) already has a "push" action encoded as part of its domain theory - this action must be updated in order to reflect the new information from the commander. This requires a change to the world model that the planner is using - the system must process information that is received via natural language (from the commander) and relay it to the system, where update methods are used in order to modify the model.

We demonstrate a run of the robotic agent through the above scenario, while it is supported by the planner. Full details of this demonstration can be found in the attached document that details a storyboard.

Planning System

The planner that we use – *SapaReplan* – is an extension of the metric-temporal planner *Sapa* (Do and Kambhampati 2002) that handles partial satisfaction planning (Benton, Do, and Kambhampati 2009) and replanning (Cushing, Benton, and Kambhampati 2008). Specifically, the planning problem is defined in terms of the initial state, and the set of goals that need to be satisfied. Actions have known (real-valued) costs. Each goal can have a reward and a penalty $\in [0, \infty]$. The reward is accrued when the goal is satisfied in the final state, while the penalty is incurred for not satisfying it. The costs, rewards and penalties are all assumed to be in the same units. The net benefit of a solution plan is defined as the sum of rewards of the goals it achieves, minus the sum of penalties of the goals it fails to achieve, and minus the sum of costs of the actions used in the plan. The use of a reward / penalty model allows our planner to model both opportunities and commitments/constraints in a uniform fashion. A goal with zero penalty is a pure opportunity, while one with zero reward is a pure commitment. A "hard" goal has finite reward but infinite penalty (and thus *must be achieved* by any plan).

The planner consists of three coupled, but distinct parts:

- Search. *SapaReplan* performs a weighted A*, forward search using *net benefit* as the optimization criterion.
- Heuristic. The heuristic used to guide the planner's search is based on well-known relaxed planning graph heuristics where, during search, relaxed solutions are found in polynomial time per state. *Sapa* uses a temporal relaxed planning graph that accounts for the durations of actions when calculating costs and finding relaxed solutions. In the partial satisfaction planning extensions, the heuristic also performs online goal selection. In essence, it solves for all goals (hard and soft) in the relaxed problem and gives a cost for reaching each of them (∞ for unreachable goals). If the cost of reaching a soft goal is greater than its reward, it removes that goal from the heuristic calculation. If the cost of reaching a hard goal is infinity, it marks a state as a dead end. Finally, the difference between the total reward and total cost of the remaining goals is calculated and used as the heuristic value.
- Monitoring / Replanning. The extensions for replanning require the use of an execution monitor, which takes updates from the human-robot team architecture (in this case). Upon receiving an update, the planner updates its knowledge of the "current state" and replans. Replanning itself is posed as a new partial satisfaction planning problem, where the initial and goal states capture the status and commitments of the current plan (Cushing, Benton, and Kambhampati 2008).

Problem Updates New sensory information, goals, or facts given by a human commander can be sent to the planner at any time, either during planning or after a plan has been output. Regardless of the originating source, the monitor listens for updates from a single source from the architecture and correspondingly modifies the planner's representation of the problem. Updates can include new objects, timed events (i.e., an addition or deletion of a fact at a particular time, or a change in a numeric value such as action cost), the addition or modification (on the deadline or reward) of a goal, and a time point to plan from. An example update is given below:

```
(:update
 :objects
    red3 - zone
 :events
    (at 125.0 (not (at red2)))
    (at red3)
    (visited red3)
 :goal (visited red4) [500] - hard
 :now 207.0)
```

All goals are on propositions from the set of boolean fluents in the problem, and there can only be one goal on any given proposition. In the default setting, goals are hard, lack deadlines and have zero reward[1]. All fields in an update specification, with the exception of ":now" (representing the time we expect to begin executing the plan), may be repeated as many times as required, or left out altogether. The intent of allowing such a flexible representation for updates is to provide for accumulation of changes to the world in one place. In the particular example provided, a new object "red3" of type "zone" is declared. In addition, three new *events* are defined, one of them with a temporal annotation

[1]Since these goals are *hard*, they can be seen as carrying an infinite penalty; i.e., failing to achieve even one such goal will result in plan failure.

that describes the time at which that event became true. A new hard goal that carries 500 units of reward is also specified, and the update concludes with the specification of the current time.

As discussed by (Cushing, Benton, and Kambhampati 2008), allowing for updates to the planning problem provides the ability to look at unexpected events in the open world as new information rather than faults to be corrected. In our setup, problem updates cause the monitor process to restart the planner (if it is running) after updating its internal problem representation.

Goal and Knowledge Revision

An important problem that the robot (and planner) must deal with is the specification of the goals that must be achieved in a given task. This goal specification may consist of the actual goals to be achieved, as well as the values of achieving such goals, and priorities and deadlines (if any) associated with these goals. The fact that the system's goals are determined and specified by the human in the loop also introduces the possibility that goals may be specified incompletely or incorrectly at the beginning of the scenario. Such a contingency mandates a need for a method via which goals, and the knowledge that is instrumental in achieving them, can be updated.

The biggest planning challenge when it comes to the problem of goal update and revision is that most state-of-the-art planning systems today assume a "closed world" (Etzioni, Golden, and Weld 1997). Specifically, planning systems expect full knowledge of the initial state, and expect up-front specification of all goals. Adapting them to handle the "open worlds" that are inherent in real-world scenarios presents many challenges. The open world manifests itself in the system's incomplete knowledge of the problem at hand; for example, in the search and report scenario, neither the human nor the robot know where the injured humans may be. Thus an immediate ramification of the open world is that goals may often be conditioned on particular facts whose truth values may be unknown at the initial state. For example, the most critical goal in the USAR scenario – reporting the location of injured humans – is conditioned on finding injured humans in the first place. In this section, we describe recent work on bridging the open nature of the world with the closed world representation of the planner that has been done in the context of the USAR problem.

Open World Quantified Goals

Open world quantified goals (OWQG) (Talamadupula et al. 2010) combine information about objects that *may be* discovered during execution with partial satisfaction aspects of the problem. Using an OWQG, the domain expert can furnish details about what new objects may be encountered through sensing and include goals that relate directly to the sensed objects. Newly discovered objects may enable the achievement of goals, granting the opportunity to pursue reward. Formally, an open world quantified goal (OWQG) is a tuple $\mathcal{Q} = \langle F, \mathcal{S}, \mathcal{P}, \mathcal{C}, \mathcal{G} \rangle$ where F and $\mathcal{S}$ are typed variables that are part of the planning problem. F belongs to the object type that $\mathcal{Q}$ is quantified over, and $\mathcal{S}$ belongs to the object type about which information is to be sensed. $\mathcal{P}$ is a predicate which ensures sensing closure for every pair $\langle f, s \rangle$ such that f is of type F and s is of type $\mathcal{S}$, and both f and s belong to the set of objects in the problem, $\mathcal{O} \in \Pi$; for this reason, we term $\mathcal{P}$ a *closure condition*. $\mathcal{C} = \bigwedge_i c_i$ is a conjunctive first-order formula where each c_i is a statement about the openness of the world with respect to the variable S. For example, `c = (in ?hu - human ?z - zone)` with `S = ?hu - human` means that c will hold for new objects of the type 'human' that are sensed. Finally $\mathcal{G}$ is a quantified goal on $\mathcal{S}$.

Newly discovered objects may enable the achievement of goals, granting the opportunity to pursue reward. For example, detecting a victim in a room will allow the robot to report the location of the victim (where reporting gives reward). Given that reward in our case is for each reported injured person, there exists a quantified goal that must be allowed partial satisfaction. In other words, the universal base, or total grounding of the quantified goal on the real world, may remain unsatisfied while its component terms may be satisfied.

As an example, we present an illustration from our scenario: the robot is directed to "report the location of all injured humans". This goal can be classified as open world, since it references objects that do not exist yet in the planner's object database; and it is quantified, since the robot's objective is to report *all* victims that it can find. In our syntax, this information is encoded as follows:

```
1 (:open
2   (forall ?z - zone
3     (sense ?hu - human
4      (looked_for ?hu ?z)
5       (and (has_property ?hu injured)
6            (in ?hu ?z))
7   (:goal (reported ?hu injured ?z)
8            [100] - soft))))
```

In the example above, line 2 denotes F, the typed variable that the goal is quantified over; line 3 contains the typed variable $\mathcal{S}$—the object to be sensed. Line 4 is the unground predicate $\mathcal{P}$ known as the closure condition (defined earlier). Lines 5 and 6 together describe the formula $\mathcal{C}$ that will hold for all objects of type $\mathcal{S}$ that are sensed. The quantified goal over $\mathcal{S}$ is defined in line 7, and line 8 indicates that it is a soft goal and has an associated reward of 100 units. Of the components that make up an open world quantified goal $\mathcal{Q}$, $\mathcal{P}$ is required[2] and F and S must be non-empty, while the others may be empty. If $\mathcal{G}$ is empty, i.e., there is no new goal to work on, the OWQG $\mathcal{Q}$ can be seen simply as additional knowledge that might help in reasoning about other goals.

Handling OWQGs in the Planning System

To handle open world quantified goals, the planner grounds the problem into the closed-world using a process similar to

[2]If $\mathcal{P}$ were allowed to be empty, the planner could not gain closure over the information it is sensing for, which will result in it directing the robot to re-sense for information that has already been sensed for.

Skolemization. More specifically, we generate *runtime objects* from the sensed variable $\mathcal{S}$ that explicitly represent the potential existence of an object to be sensed. These objects are marked as system generated runtime objects. Given an OWQG $\mathcal{Q} = \langle F, \mathcal{S}, \mathcal{P}, \mathcal{C}, \mathcal{G} \rangle$, one can look at $\mathcal{S}$ as a Skolem function of F, and runtime objects as Skolem entities that substitute for the function. Runtime objects are then added to the problem and ground into the closure condition $\mathcal{P}$, the conjunctive formula $\mathcal{C}$, and the open world quantified goal $\mathcal{G}$. Runtime objects substitute for the existence of S dependent upon the variable F. The facts generated by following this process over $\mathcal{C}$ are included in the set of facts in the problem through the problem update process. The goals generated by $\mathcal{G}$ are similarly added. This process is repeated for every new object that F may instantiate.

We treat $\mathcal{P}$ as an *optimistic closure condition*, meaning a particular state of the world is considered closed once the ground closure condition is true. On every update the ground closure conditions are checked and if true the facts in the corresponding ground values from $\mathcal{C}$ and $\mathcal{G}$ are removed from the problem. By planning over this representation, we provide a plan that is executable given the planning system's current representation of the world until new information can be discovered (via a sensing action returning the closure condition). The idea is that the system is interleaving planning and execution in a manner that moves the robot towards rewarding goals by generating an optimistic view of the true state of the world.

Conclusion

In this paper, we proposed the demonstration of a robotic agent being guided through a complex search and rescue scenario by a planning system. We discussed the motivation behind using a planner in such a task, and described the scenario in detail. We then gave a brief overview of the planning system in use, and the procedure via which the planner's world knowledge is updated. Following this, we visit the problem of updates to the agent's goals, and describe a construct that enables the planner to deal with such updates. We conclude with an explanation of how this construct is accomodated into the planning system.

References

Benton, J.; Do, M.; and Kambhampati, S. 2009. Anytime heuristic search for partial satisfaction planning. *Artificial Intelligence* 173(5-6):562–592.

Cushing, W.; Benton, J.; and Kambhampati, S. 2008. Replanning as a Deliberative Re-selection of Objectives. *Arizona State University CSE Department TR*.

Do, M., and Kambhampati, S. 2002. Planning graph-based heuristics for cost-sensitive temporal planning. In *Proceedings of AIPS*, volume 2.

Etzioni, O.; Golden, K.; and Weld, D. S. 1997. Sound and efficient closed-world reasoning for planning. *AIJ* 89(1-2):113–148.

Talamadupula, K.; Benton, J.; Schermerhorn, P.; Scheutz, M.; and Kambhampati, S. 2010. Integrating a Closed-World Planner with an Open-World Robot. In *AAAI 2010*.