EU FP7 CogX
ICT-215181
May 1 2008 (52months)

DR 4.2:
Planning for Cognitive Robots

Michael Brenner, Richard Dearden, Moritz Göbelbecker, Charles Gretton, Patrick Eyerich, Thomas Keller, Bernhard Nebel

Albert-Ludwigs-Universität Freiburg, University of Birmingham
⟨brenner@informatik.uni-freiburg.de⟩

Due date of deliverable:	31 July 2010
Actual submission date:	31 July 2010
Lead partner:	ALU
Revision:	v2
Dissemination level:	PU

Planning is a crucial capability for autonomous cognitive agents. In CogX, we aim at developing intelligent robots that act deliberately in realistic dynamic environments where knowledge about the world is fragmentary and prone to become obsolete quickly. Planning in such environments is very hard, but nevertheless the robot needs to decide what to do next in close-to real time. This report describes our approach to meeting these requirements. We have constructed a system that can switch between a fast continual planner and a more computationally expensive decision-theoretic planner. The idea is for the fast planner to plan as if any variable in the world could be observed directly. Then each time during execution of this plan the robot observes the value of some state variable, the decision-theoretic planner is called to build a plan to become as sure as possible of the value of that variable. If the value is not the one the fast planner assumed, then replanning occurs to find a plan to cope with the newly discovered facts. We present a prototype of this approach, and examine some of the tradeoffs inherent in different designs.

Executive Summary

The objective of Workpackage 4 is to develop techniques that enable intelligent agents to act deliberately in highly dynamic environments. In such environments knowledge about the world is fragmentary and prone to become obsolete quickly. Deliberate action planning in such environments transcends the standard view of planning as "one-shot" problem solving. Instead, agents must be able to build plans that include both physical and information-gathering actions. These actions may be stochastic or non-deterministic, and the system must reason about their possible outcomes. However, since the agents developed in CogX are robots acting in the real world in real time the planning system must make decisions quickly. These are competing requirements. We have therefore proposed to develop, during the course of the project, a hybrid planner that can switch between an accurate, yet computationally expensive decision-theoretic planner and a fast classical planner.

In the second year of CogX we have worked primarily on how to take advantage of the strengths and avoid the weaknesses of each planner. While we proposed in the first year to do domain analysis to find out where each planner should be deployed, we have changed our approach to one we believe will be much more effective. In our new switching planner design we use the decision-theoretic planner to resolve uncertainty for the continual planner. That is, the continual planner generates a plan that assumes it can observe any domain variable at any time. Then, when a domain variable is actually observed, the decision-theoretic planner is run to determine, as best it can, the true value of that variable, before control switches back to the continual planner.

Role of Planning in CogX

Planning is a crucial capability for any cognitive agent, because it enables it to act autonomously: rather than just executing pre-defined scripts, a planning agent can devise its own solutions for the goals it is given or which it develops. In the CogX project, planning has an additional important role in detecting and filling gaps in knowledge: information-gathering actions, e.g. active visual search or asking a question, will be planned whenever there is a knowledge gap crucial for achieving a goal.

Contribution to the CogX scenarios and prototypes

The planners developed in this workpackage are responsible for all behaviour-related decisions in the CogX prototypes Dora and George. In contrast to Year 1, where only the Continual Planner was used, we have now developed

a decision-theoretic planner, too, which is integrated into the new switching planner. Additionally, planning is now also used for goal selection from within the Motivation subarchitecture.

1 Tasks, objectives, results

1.1 Planned work

WP4 focuses on planning and decision making. During all its activities, the robot constantly has goals either requested of it through interaction, or generated from its own internal motivations. To turn these goals into behaviours that the robot will execute, planning is required to determine how the goals can be accomplished efficiently.

Real-time performance is a major concern for an interactive robot. A robot that takes a long time to respond to a dialogue act, or to decide what to do is essentially useless as no human user will be willing to interact with it. This puts serious resource constraints on planning. To address this, in the proposed work we identified a switching symbolic/decision-theoretic planner as a desirable goal for the project. This was expressed in Task 4.1.

Task 4.1: A switching symbolic/decision-theoretic planner. *In this task we will look at how to combine these two approaches (symbolic and decision theoretic) into the switching planner discussed above. The result should be a planner that can do limited reasoning about belief states (the representation in terms of epistemic operators is much less expressive than a full probabilistic belief-state representation) but still make good decisions, and that will operate in close-to real-time.*

In addition, in the second year we have begun task 4.2, which involves planning of information gathering actions. The Dora Year 2 scenario involves identifying rooms, so this is a prime example of this task.

Task 4.2: General planning of information gathering and dialogue actions. *The symbolic planner developed in Task 4.1 is limited in that it uses epistemic operators to represent beliefs, so it can only represent that a fact is known to be true, known to be false, or unknown. Better plans can be achieved by representing a much richer set of beliefs, for example by using probabilistic belief states. This allows the system to reason about the most likely states given its current knowledge, so it can for example begin driving towards the kitchen when sent to look for the cornflakes because its a-priori belief is that they are most likely to be in the kitchen. In Task 4.2 we will extend the planning system to allow arbitrary belief states to be reasoned about. The aim is to produce a planner capable of planning over arbitrary belief states, but specialised for the requirements of our domain.*

Following this agenda, in this deliverable we report on our development of a switching planner based on the two base planner systems, the common language used by both planners, and how the planning system fits into the overall architecture.

1.2 Actual work performed

1.2.1 Symbolic Continual Planner

Automated planning for dynamic real-world environments is challenging for deterministic AI Planning approaches in several respects: Due to noisy sensor information an agent may not be able to accurately model the true state of the world; actions might result in different states than foreseen, as their execution might fail or produce effects that cannot be modelled; or subproblems may be present that cannot be solved on a symbolic level. Work carried out at ALU Freiburg tries to deal with these difficulties while preserving the computational advantages of deterministic planning over non-deterministic approaches (e.g., the decision-theoretic planner presented in Section 1.2.2). We do this by adopting a *Proactive Continual Planning (PCP)* approach, i.e., the planner actively tries to execute actions early and observe previously unknown parts of its environment in order to avoid having to plan for all possible contingencies in advance [1].

In the first year of CogX, the Continual Planning framework was designed and implemented within the Dora system. We developed a specific base planner to be used for PCP called Temporal Fast Downward (TFD), which both works with arbitrary optimization metrics including temporal information and produces plans in an anytime fashion [2]. Finally, we proposed the use of *semantic attachments* to compute the truth values of propositions with external modules that may contain arbitrary calculations at runtime of the planner [3]. In the second year of CogX we developed further extensions aimed to enhance our robot's capabiliy to act in dynamic environment, especially ones where it must interact with humans. In particular we have developed

- an approach for finding explanations for planning failures so that the human can at least understand what is wrong with the robot's information state
- a representation and algorithm for describing plans involving multiple agents and goals, that can be used to reason about how to initiate collaboration, e.g., with a human who can open doors for the robot
- a representation (and planner support) for intermediate goals and preferences in a planning problem, so that human users or the Motivation subarchitecture can describe the desired course of action more flexibly
- an initial approach to incorporating probabilistic information into the continual planning process, applied to a path planning problem under partial observability

Since information a robot gains about the world is often uncertain or incomplete it often happens that the planner is unable to come up with

a plan given by a human user at all. This might for example happen if the robot is told to put away a cup the vision subarchitecture was unable to detect. In such a case, the human user might like to know why the agent couldn't find a solution. This is a hard question and not one that a planner normally can answer, because this would require meta-reasoning about its own planning process. However, it is a question that both users and domain designers (for debugging purposes) often raise. For this purpose, we introduced the concept of *excuses*, explanations of why the planner is not able to find a plan [4]. This enables the agent to ask for specific help from a human rather than merely admitting its own incompetence – in the example above, it might for instance tell the instructor that it doesn't see any cups and ask for a hint where to start looking for one.

Furthermore, we have expanded the system's functionality in two ways: We introduced *intermediate goals* and *preferences*, which can appear as part of the goal formula or, in the latter case, in preconditions. Intermediate goals describe a property that has to be obtained sometime during plan execution, but not necessarily in the goal state. Preferences describe a property of a state that is desired to be valid but not necessarily required, leading to a penalty that's added to the plan's cost in the case it is violated.

In order to plan for interactions with others, we had previously suggested a continual collaborative multiagent planning approach [1]. In year 2, we have extended this approach to general multiagent scenarios where agents are not necessarily collaborative and, in particular, where collaboration must first be initiated and negotiated. As a first step we have implemented this approach, not on the robot architecture, but in a multiagent simulation. It uses the same algorithm, but applies for a different task, the generation of story plots [5].

We have also made a first step in extending Continual Planning to stochastic domains. Currently, our PCP algorithm is optimistic about being able to gather information it is still missing in the future and to be able to fill the gaps in its plan then. This, of course, is not always the case. We have therefore, for a limited problem of probabilistic planning (the Canadian Traveler's Problem) developed a new algorithm based on Monte Carlo sampling [6]. In the remainder of the project, we plan to extend this approach to general planning tasks, thereby bridging the gap between the Continual and Decision-Theoretic Planning approaches.

Relevant annexes:

- Annex 2.1 describes our approach to explaining planning failures and the concept of "excuses".
- Annex 2.2 is a "challenge paper" describing a problem typically faced by temporal planners.

- Annex 2.3 describes our Monte Carlo method for a probabilistic path planning.
- Annex 2.4 describes Continual Multiagent Planning with an application to generating dynamic story plots.
- Annex 2.5 shows an application of our base planner, Temporal Fast Downward with semantic attachments, to a multi-robot exploration scenario.

1.2.2 Decision-Theoretic Planner

Unlike the symbolic planner, which is based around an existing planner, the decision-theoretic planner is being developed as part of the task. As a representation, we have chosen *partially observable Markov decision processes* (POMDPs) as they constitute a reasonably general representation of uncertainty. Because of that generality, state-of-the-art domain independent solution procedures are quite limited, only able to solve quite (unrealistically) small problems. Indeed, there are no known general techniques that scale well as we increase the size and difficulty of problems.

For our purposes, a POMDP is a six-tuple $\langle \mathcal{S}, \mathcal{A}, \mathsf{Pr}, \mathsf{R}, \mathbb{O}, v \rangle$. Here, $\mathcal{S}$, $\mathcal{A}$, Pr, and R are states, actions, state-transition function, and reward function, respectively—they provide a Markov Decision Process (MDP)-based specification of the underlying world state, dynamics, and reward. $\mathbb{O}$ is a set of observations. For each $s \in \mathcal{S}$ and action $a \in \mathcal{A}$, an observation $o \in \mathbb{O}$ is generated independently according to some probability distribution $v(s, a)$. We denote $v_o(s, a)$ the probability of getting observation o in state s.

Finite State Controller Based Solution Procedure: The optimal solution to a finite-horizon POMDP problem can be expressed as a policy $\mu : \mathbb{O}^* \rightarrow P_{\mathcal{A}}$ where $\mu_a(o_0, .., o_t)$ is the probability that we execute action a given observation history $o_0, .., o_t$.[1] A finite-state controller (FSC) is a more useful policy representation mechanism in the case that the robot has unbounded interactions with the environment modelled by the POMDP at hand. Therefore an approach to solving POMDPs is to build a *finite-state controller* (FSC) for a POMDP, with that controller representing a policy. A FSC is a three-tuple $\langle \mathcal{N}, \psi, \eta \rangle$ where: $n \in \mathcal{N}$ is a set of nodes, $\psi_n(a) = P(a|n)$, and $\eta_n(a, o, n') = P(n'|n, a, o)$. The value of state s at node n of the FSC for a given POMDP is:

$$V_n(s) = \sum_{a \in \mathcal{A}} \psi_n(a) \mathsf{R}(s, a) + \beta \sum_{a,o,s',n'} \eta_n(a, o, n') \mathsf{Pr}(s, a, s') v_o(s', a) V_{n'}(s') \quad (1)$$

[1] Such a policy can oftentimes be compactly represented as a tree or algebraic decision diagram (ADD).

If b is a POMDP belief state—i.e, $b(s)$ gives the probability that the robot is in state s—then the value of b according to the FSC is:

$$V_{\mathtt{FSC}}(b) = \max_{n \in \mathcal{N}} \sum_{s \in \mathbb{S}} b(s) V_n(s) \tag{2}$$

There is a small amount of work that considers POMDP solution procedures based on a FSC representation [7, 8, 9, 10, 11, 12, 13]. Generally, such a procedure proceeds in two steps: (1) *evaluation*, and (2) *improvement*. The evaluation step consists in fixing η_n and ψ_n, and then solving the linear system described in Equation 1. For improvement, one technique is to solve the following linear program at controller nodes $\mathcal{N}$.

$$\begin{array}{ll}
\texttt{maximise} & \epsilon \\
\quad \texttt{s.t.} & \forall s \in \mathcal{S}, \forall a \in \mathcal{A} \\
& V_n(s) + \epsilon \leq \sum_a \psi_n(a) \mathsf{R}(s,a) + \\
& \quad \beta \sum_{a,o,n',s'} \eta_n(a,o,n') \mathsf{Pr}(s,a,s') v_o(s',a) V_{n'}(s') \\
& \sum_{a \in \mathcal{A}} \psi_n(a) = 1 \\
& \forall a \in \mathcal{A}, \forall o \in \mathbb{O} \\
& \sum_{n' \in \mathcal{N}} \eta_n(a,o,n') = \psi_n(a) \\
& \psi_n \texttt{ and } \eta_n \texttt{ terms are positive or zero.}
\end{array}$$

Above, for each node the LP variables include ϵ,[2] η_n, and ψ_n. The $V_n(s)$ terms are fixed according to the *evaluation* step. Policy improvement consists in solving the above LP at controller nodes until no further improvement is possible. At this point further improvement might be achieved if a new node is added to the controller. One strategy here is to copy any node where the backed up value of the *tangent belief state* – i.e. belief state at which the controller node yields the smallest expected reward – is better than the *evaluation* step estimates. The copied node is altered to prescribe the action that is greedy according to the backed up value function. Here, the backed-up value $V(b)$ at a belief state b has the following form.

$$V(b) \leftarrow \max_{a \in \mathcal{A}} \{ \mathsf{R}(b,a) + \beta \sum_{o \in \mathbb{O}} v_o(b,a) V_{\mathtt{FSC}}(b_o^a) \} \tag{3}$$

The solution procedure as outlined above iteratively converges to an ϵ-optimal solution for the problem at hand.

Online Solution Procedures: An exact implementation of the policy-iteration procedure outlined in the previous section is only useful for computing good policies for small POMDPs —with hundreds of states— and is also computationally expensive in comparisons with state-of-the-art online

[2] Note, each LP has a different ϵ.

POMDP solution procedures [14]. Online techniques have been demonstrated to be useful in applications of planning in AI and robotics. A few recent examples come from dialogue management [15], visual search [16], automated vent-mapping on the sea floor [17], and effective navigation of probabilistic roadmaps [18]. Here we briefly summarise the ancestry and homology of these approaches.

Online techniques can usually be described and implemented as a forward-search of the *information-state* space. The algorithms build a search tree (graph) where nodes correspond to belief states, the root (starting) node corresponding to the starting-state distribution. Search edges are directed, and labelled with an action, observation, and a probability value. Where there is a directed edge from node b to b' labelled with action a, observation o, and value p, then with probability p we arrive at beliefs b' when we execute a at b and receive observation o. As the search progresses it labels nodes with an optimal, or good, policy to execute at that node, along with an estimate of the lower and upper bound of the expected utility of the corresponding belief state.

Examples of online techniques include: (1) the reactive entropy reduction approach; i.e. search greedily toward successive belief states with the lowest entropy, (2) fixed-horizon *information lookahead*; i.e. solve the contingent propositional probabilistic planning problem supposing we can act for a small number of steps, (3) branch-and-bound techniques, the LAO* [19] versions of those, and their sampled counterparts [20, 21].

Progress: We have an implementation of FSC evaluation and a simple improvement procedure. We intend to apply this work directly in planning for dialogue management in our project scenarios. Our current focus is on implementing an efficient online procedure to target the many distinct small finite-horizon sensing problems that are posed by the project scenarios. We are in the process of implementing a generic online branch-and-bound procedure that supports loop detection and exploitation in the sense of LAO*. Our online procedure uses a *blind-policy* (or otherwise a given FSC) in order to compute a *lower bound* on the expected value of search nodes. The MDP approximation is used to provide an *upper bound* and heuristic [3] for the forward-search.

Relevant annexes:

- Annex 2.6 describes how to encode serial planning tasks with action costs as partially weighted MaxSAT problems. It presents an efficient backtracking MaxSAT procedure (implemented in C++) that computes optimal solutions to such problems. That same procedure can

[3] When used heuristically, the MDP approximation is sometimes called the *certainty equivalent* heuristic.

compute optimal serial solutions to fixed horizon POMDPs, or propositional probabilistic planning problems. In particular, that Annex describes how to exploit our encoding and procedure to compute globally optimal solutions to propositional planning problems with action costs.

1.2.3 Switching Planner

The original concept for the switching planner, as articulated in the Work Package description was of a planner that chose between classical (PCP) and decision-theoretic planning depending on the problem and on the potential benefits of each. The research challenge—apart from building the two planners—was to select a computationally inexpensive method to choose between them. Our work this year has concentrated on this problem. However, in the course of our investigations it has become clear that in a world with partial observability (that is, where we can't know with certainty the values of all state variables) the plans generated by PCP (see Section 1.2.1) aren't necessarily executable at all as they rely on being able to determine the values of variables in assertions. Thus the switching planner work has concentrated on solving this problem by using the decision-theoretic planner to generate plans to determine the value of variables that appear in assertions in the PCP plan. This is one situation in which switching is necessary to produce a plan that can be successfully executed. There are other situations where it may be desirable for plan quality reasons to switch from PCP to the decision-theoretic planner, but we have not yet investigated in any detail how to detect these.

In situations with partial observability, the agent has at all times a belief state which represents its current best estimate of the state of the world, but which may be quite uncertain about some state variables. As the agent acts in the world, this belief state is updated to reflect the new evidence observed. As we said in Section 1.2.2, this kind of problem is typically represented as a POMDP. When PCP builds a plan, it assumes that it can determine the value of any system variable at execution time. This is represented in the plan through the use of assertions. When an assertion appears in the plan (for example, when the robot asserts that the room it is in is a kitchen), PCP assumes that the true value of the variable is determined, and either execution continues if the true value matches the assertion, or replanning is triggered. Since determining the true values of these variables involves performing information-gathering actions to gain evidence about the variable, and may be a non-trivial problem, the switching planner we have built uses these assertions to trigger decision-theoretic planning—the idea is that the decision-theoretic planner is used to generate a plan to determine the value of the variable.

The overall architecture of the switching planner is given in Figure 1.

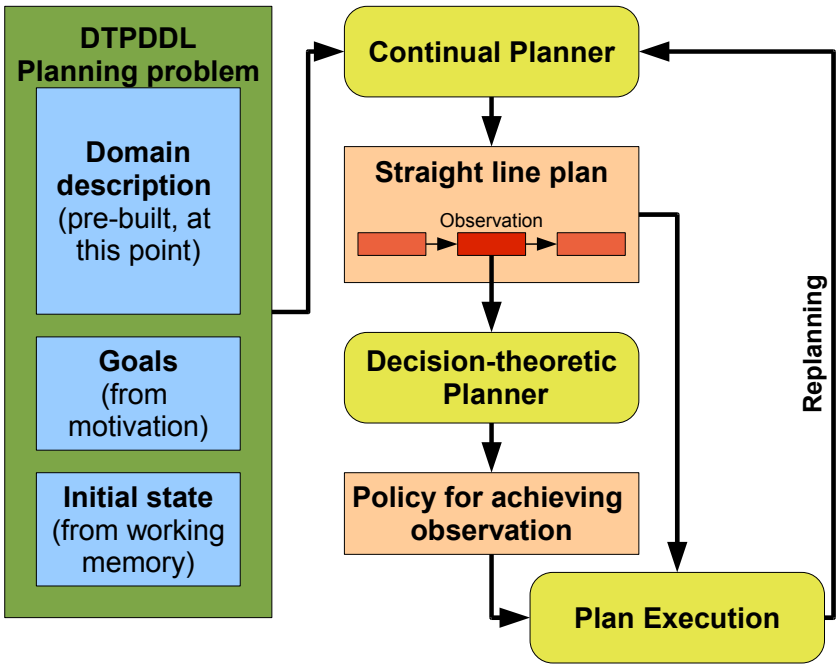


Figure 1: The architecture of the switching planner.

The input is a problem specification in the DTPDDL planning specification language we developed in Year 1 (DTPDDL and a representation of the Dora problem domain in it are given in Delverable 1.2, Section 7.4.1, so will not be repeated here). The input problem is made up of three parts:

- A domain definition that specifies the actions, which at present is pre-built, but which in the future will be learned from experience.
- An initial state that comes from working memory (specifically the binder).
- A goal or goals, which come from the motivation subarchitecture.

When the planner is given a problem to work on, the DTPDDL representation of the planning problem is translated into a form that PCP can plan with, and PCP generates a plan. This plan contains no branching points, but instead it contains points where domain variables are observed and a particular value is assumed. Thus we can think of it as a branching plan—with branches that depend on the values of domain variables at execution time—where only one branch at each branch point has actually been planned for. If a different branch is needed during execution, then the planner may be called again to replan for that branch.

During execution of the PCP plan, when a domain variable is observed, the plan executer needs a way to determine the value of that variable. When

the value isn't known with certainty, then the decision-theoretic planner is called to determine the value. It is given the same DTPDDL domain definition and the current initial state, but the goal is to determine with sufficient confidence the value of the variable. The decision-theoretic planner produces a plan for this very small problem, which can then be executed to find out the value. At that point, the plan executor can either continue with the PCP plan, or if necessary trigger PCP to replan.

The idea of this approach is that the decision-theoretic planner is only called on relatively simple planning problems for which the set of relevant actions can be restricted (only actions which produce observations of the variable of interest are directly relevant). This keeps the state and action space small, and should mean the problems have quite short plans, so the computational load is kept to a minimum.

Building these observation plans for variables in the PCP plan turns out to be the most critical scenario where the switching planner must use the decision-theoretic planner. Unfortunately, it isn't particularly suited to the FSC-based POMDP planner we had been developing. This is the reason for the construction of a LAO* POMDP planner as reported in Sectionsec:dt-planner. This work is almost complete as of July 2010 and is the last part of the switching planner that is required before it can be applied on real domains. More details of the switching planner can be found in Annex 2.7.

Relevant annexes:

- Annex 2.7 is an early draft of a paper on the switching planner that we are planning to submit to ICAPS 2011. It describes in detail the architecture of the switching planner and the way it might work in practice. There is no experimental evaluation at present as that is work in progress.

1.2.4 Relationship between the Planner and the Overall Architecture

In order to create the planning problems for a robotic system, the planner needs to find out what the system is supposed to do (goal generation) and what the current state of the system is (state generation). Additionally, the planner may be able to use *control knowledge*, domain specific knowledge that may enable the planner to find better plans. This information is not part of the planning components itself but is distributed over the entire system. Figure 2 shows the interactions of the planning components with the rest of the CogX architecture. The motivation component is responsible for generating goals while the Binding and Default subarchitectures store information about the system's state and the robot's background knowledge.

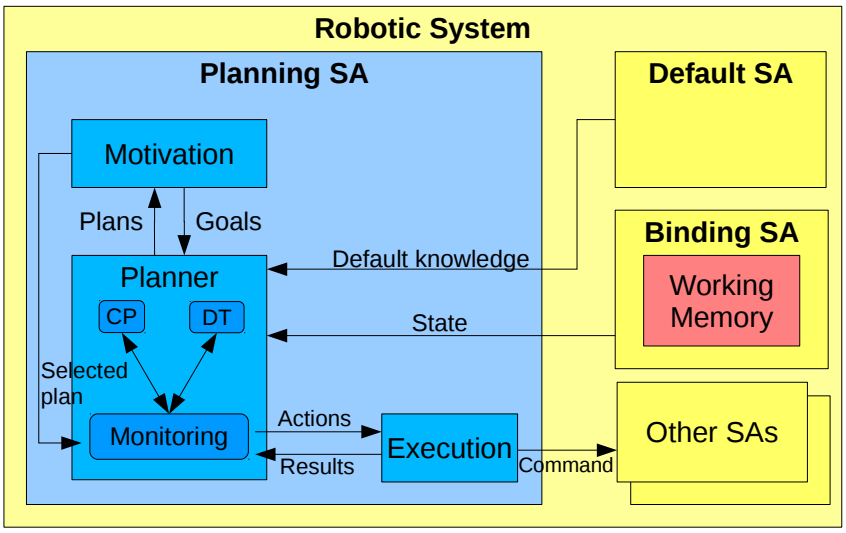


Figure 2: The relationship between a planner and the rest of the architecture.

When a plan is generated, the execution component is responsible for translating planner actions into subarchitecture specific commands and informing the planner of the results.

Knowledge Representation: The planning subarchitecture has three sources of information to create the planning problem. First, we have the planning domain description. It describes the actions the robot can perform, their preconditions and outcomes. It also defines the predicates and functions that the planners will use to describe their internal planning states. Second, there is *instance knowledge* that is comprised of the robot's perceptions and information given to it by other agents (e.g. humans). Finally there is *default knowledge* that describes general information about the robot's environment. For example "The cornflakes box could be in room 5 or room 6" is instance knowledge while "Cornflakes boxes are usually found in kitchens" is default knowledge. Both need to be taken into account in order to create high quality plans.

In the CogX system, the planning domain is provided as an external, task specific input. Even though the domain is largely fixed, the planning system will modify the planning domain in order to generate input to the different planners, or to incorporate default knowledge (see below). In the future we also want to allow subarchitectures to create new actions and add them to the planning domain at runtime and add limited learning of actions. Instance knowledge is represented as a set of *Beliefs* that are stored in the working memory of the Binding SA (the "Binder"). Each belief refers to a real-world entity and contains distributions over features of this entity.

Default knowledge is stored inside a separate subarchitecture, Default SA, in the form of a Bayesian network. Default SA also provides services to other subarchitectures to infer features of beliefs given the prior beliefs on the binder.

The easiest way to take default knowledge into account for planning is using these services to compute the posterior beliefs and use those to create the initial planning state. While this approach gives us a more informed initial state (e.g. we can infer that the cornflakes box is more likely in a room labeled "kitchen") the planner doesn't have access to the default knowledge itself. In cases where there is little prior information that Default SA can use, taking default knowledge into account during planning would be desirable. So in a case where there is no kitchen, the planner could include actions to find the kitchen first before trying to find the cornflakes box.

For use inside the continual planner, default knowledge may be used as a kind of control knowledge by incorporating it into the planning domain itself. One way to do this is by augmenting existing observation actions with *preference conditions* that are extracted from the default knowledge. For example, an action to search for objects in a room might include a preference condition that the room label indicates that it is likely to find the object there.

For the decision theoretic planner, we use the prior belief state together with the Bayesian network to create a distribution over possible initial states.

Planning and Motivation: Together with the planner, the motivation component controls the high level behaviour of the robotic system. While the planner decides how to achieve the system's goals, motivation decides *what* these goal should be in the first place [22].

When there exists more than one goal at a time there needs to be a mechanism to decide which goals to pursue. For motivation to make informed choices about the goal selection, it needs information about the *costs* of achieving a certain goal – information only the planner can provide as these costs generally depend on the plan.

There are two basic ways to do goal selection: The first method is to leave goal selection solely to the motivation component. Motivation would ask the planner for a plan for each goal in order to get a cost esimate for the goals. Using those estimates and the priority or gain for each goal, motivation would select a set of the "best" goals and send it to the planner to find a plan that achieves them. The obvious deficency of this method is that it ignores interactions between goals. In reality though, the costs of achieving two goals might be either lower (by exploiting synergies) or higher (because of conflicts) than the sum of both costs. Determining the costs for achieving each subset of goals would be a possible solution but is obviously not feasible for more than a few goals.

Thus it is appealing to leave the decision on which goals to satisfy to the planner. In the implementation of *oversubscription planning* using "soft goals" (non-mandatory goals), each goal is augmented with a penalty that would be incurred if a plan doesn't achieve it. This mechanism allows motivation to set the maximal costs for a given goal. If the costs of achieving it surpass this maximum, it would be cheaper to incur the penalty instead of trying to achieve the goal. While this method accounts for the interaction between goals, oversubscription planning for many goals may become quite slow. So in practice we may want to employ a combination of both methods.

To do this more informed goal selection, a stronger integration between planning and motivation is required. Instead of posting a goal to the planner and executing the result, in the new architecture motivation is able to post *planning requests* which can be processed by the planner in parallel, evaluate the resulting plans and decide then if a plan should be executed.

1.3 Relation to the state-of-the-art

1.3.1 Diagnosis of Planning Failures

Detecting execution failures and recovering from them is a typical task for any robotic system. In plan-based agent architectures this can be done by plan monitoring and plan repair (in its most simple variant this can be achieved by full replanning) [1]. A problem that is much harder to deal with than an execution failure is a failure to find a plan. In CogX, the robot might for instance be located in a living room with a locked door to the kitchen, and receive the order to tidy up the kitchen table. As our agent is unable to open the locked door, it is unable to come up with a plan. Better than merely admitting its incompetence would be if the robot could provide a good excuse – an explanation of why it was not able to find a plan. In this example, the robot would recognize that if the kitchen door were unlocked it could achieve its goals.

Annex 2.1 describes our approach to explaining planning failures and the concept of "excuses". We are not aware of other work in the area of AI planning that addresses the problem of explaining why a goal cannot be reached. However, there is some overlap with abduction (a term introduced by the philosopher Peirce), counterfactual reasoning [23], belief revision [24], and consistency-based diagnosis [25]. All these frameworks deal with identifying a set of propositions or beliefs that either lead to inconsistencies or permit to deduce an observation. There are parallels to our notions of acceptable, good and perfect approaches in these fields [26]. The main difference to the logic-based frameworks is that in our case there is no propositional or first-order background theory. Instead, we have a set of operators that allows us to transform states. There has been some work in the somewhat related field of error diagnosis, though. For instance, Howe and Cohen describe how

erroneous plans can be analysed in order to debug a planning system [27].

1.3.2 Continual Planning

Planning in dynamic, partially observable environments is usually modelled as a *conformant*, *contingent* or *probabilistic* planning problem. These approaches compute conditional plans or policies for the possible contingencies such that the agent can react adequately when faced with them. Unfortunately, this increased flexibility comes at the cost of being computationally much harder than classical planning [28, 29]. Thus, these approaches scale badly in dynamic multiagent environments with large numbers of unobservable features and exogenous events. Therefore, we employ a different approach: Instead of considering many possible futures in advance, an agent may execute parts of its plan in order to gather additional information, thereby reducing the number of possible contingencies that it has to take into account for the remaining planning.

This technique of interleaving planning, plan execution and execution monitoring is called Continual Planning (CP). CP is often advocated as a practical approach to planning in dynamic or incompletely known domains. Yet, previously, only few principled approaches to CP have been described [30, 31]. Our own CP approach is based on the idea of *proactive knowledge-gathering*: instead of planning for all possible contingencies, agents try to learn more about the state of the world directly [1]. In order to enable agents to reason about how they can gather additional knowledge it is necessary to explicitly model the agents' beliefs as well as their sensing capabilities as part of their formal planning domain [32, 33, 31, 34].

When planning for interaction with other agents, a robot must reason about their states of mind, too. To model the beliefs of different agents (and their reasoning about each other) *epistemic* modalities must be integrated into the planning representation [35]. Additionally, similarly to *BDI* models of multiagent cooperation, it must be able to model the desires and intentions of different agents [36]. Proactive information gathering, as needed for CP, is not limited to sensing in a multiagent setting; dialogue is an essential means to constrain the possible futures during CP as well [37].

1.3.3 POMDP State-of-the-art

In recent times there have been very promising developments towards practical and efficient algorithms for solving POMDPs. Although the majority of this work has developed approximate (resp. optimal) solution algorithms, the community has also made headway in the direction of practical optimal algorithms. Whatever is sought in terms of the quality of the solution, proposed techniques are either based on Sondik's adaptations [38, 39] of Howard's value and policy iteration [40], or reinforcement learning [41].

Value iteration algorithms work by iteratively improving a value function. Theoretically the optimal value function can always be expressed as a piecewise linear and convex mapping from belief states to the reals. Thus, the most popular representations are vector-based, however for approximate algorithms piecewise constant – also called *tabular* – representations have shown promise [20]. Policy iteration represents the control strategy directly. Here a given initial controller is iteratively improved by performing policy evaluation and policy improvement operations. Although not a significant focus of our work, many reinforcement learning approaches for fully observable MDPs have occasionally demonstrated excellent performance in settings that feature partial observability [42, 43, 44, 21]. Most notably, FPG [21] is a reinforcement learning based planning system that achieved first-place in the uncertainty track of the 2006 International Planning Competition.

There is an enormous literature proposing a vast collection of approaches for reasoning about POMDPs. We focus on the state-of-the-art in automated planning. Approaches here fall broadly into three classes: (1) point-based value iteration [45, 46, 47, 48], (2) heuristic search with a tabular or vector-based value function presentation [49, 50, 51, 20], and (3) bounded policy iteration [7, 10, 52, 12, 13]. Value-based approaches have two advantages over their policy-based counterparts. First, the state-of-the-art is universally approximate or online, targeting a problem instance with a specific starting state distribution. Consequently they are significantly faster at computing a good solution for a given problem. The second advantage is very pragmatic. Good implementations of many of the approaches are available off-the-shelf. These include Blai Bonet and Héctor Geffner's RTDP-Bel, Matthijs Spaan's Perseus, Pascal Poupart's symbolic variant of Perseus, and Trey Smith's ZMDP.

Policy-based approaches have two advantages that we consider to be fundamentally important given the CogX goals of self-*understanding* and self-*extension*. First, unlike value-based approaches, there is no need for expensive belief propagation during policy execution[4] – i.e., plan execution is relatively cheap. Second, a policy can be *evaluated* and *executed* over *multiple* problem instances – i.e., the policy (resp. its value) is not parametrised by problem states, but rather by observations and actions. Key to an agent's *understanding* of a problem is the language over perceptions and actions that it has inferred to be important for acting well in its environment. A controller derived using policy iteration is represented compactly in terms of a small graph capturing the observation-action history distinctions necessary for rewards to be accumulated and goals to be achieved. Indeed, the controller and domain model correspond more-or-less exactly to the agent's *understanding* of the task at hand. In more detail, given a controller and model, the robot can directly answer queries – sourced introspectively or

[4]Intractable for the optimal case [53].

otherwise – about: (1) how well it expects to perform with respect to certain objectives, (2) how well it can perform given a specific change to the domain model, and (3) performance improvements that are obtained if we extend the controller or modify its parameters. The second advantage can be cast in terms of self-extension. Indeed, *extension* occurs where interleaved evaluation and improvement of a given policy is undertaken to accommodate new or changed observations, actions, rewards, and goals – i.e., to adapt a controller for a particular problem instance.

1.3.4 Switching Planner

The problem the switching planner is trying to solve is essentially the same as the POMDP planners we discussed in Section 1.2.2 so all the work mentioned in Section 1.3.3 is relevant. The key difference in the switching planner is that we are trying to reduce the computational requirements by using PCP to perform as much of the planning as possible, and only using a POMDP planner to reduce the uncertainty in our belief state when necessary.

The other approach that solves similar kinds of planning problems is to use classical planners to generate plans with branches and loops in them. These do not resemble our switching planner approach except in the sense that they try to solve decision-theoretic problems using classical planners to reduce computational demands. Examples of this include the limited contingency planning work of [54], although this is operating in a completely observable domain and cannot produce plans with loops, and C-Buridan [55], which solves partially observable problems, but seeks to maximise the probability of goal achievement rather than plan quality.

As we mention in Annex 2.7, another way to think of the switching planner is as the top two tiers of a three-tiered robotic architecture such as 3T [56] (see the Architectures chapter in [57] for a survey of such architectures). The idea of these is to use a classical planner at the top level for decision making. To reduce the uncertainty inherent in robotic domains, the actions from the high-level planner are decomposed into sequences of low-level control. Thus a high-level action GOTO(HALL) decomposes into a sequence of wheel rotations interspersed with checks to see if the hall has been reached. The architecture's three tiers consist of the classical planner at the top, the control rules at the bottom, and in the middle, a set of skills or *reactive action plans* (RAPs) that provide alternative ways to translate the high-level actions into control. Our switching planner can be thought of in the same way, with the actions from the other subarchitectures as the low level, PCP at the high level, and the RAPs constructed on the fly by the decision-theoretic planner.

References

[1] M. Brenner and B. Nebel. Continual planning and acting in dynamic multiagent environments. *JAAMAS*, 19(3):297–331, 2009.

[2] Patrick Eyerich, Robert Mattmüller, and Gabriele Röger. Using the context-enhanced additive heuristic for temporal and numeric planning. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 2009.

[3] Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Trüg, Michael Brenner, and Bernhard Nebel. Semantic attachments for domain-independent planning systems. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 2009.

[4] Moritz Göbelbecker, Thomas Keller, Patrick Eyerich, Michael Brenner, and Bernhard Nebel. Coming up with good excuses: What to do when no plan can be found. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press, may 2010.

[5] Michael Brenner. Creating dynamic story plots with continual multiagent planning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Press, july 2010.

[6] Patrick Eyerich, Thomas Keller, and Malte Helmert. High-quality policies for the canadian traveler's problem. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Press, july 2010.

[7] Eric A. Hansen. Solving POMDPs by searching in policy space. In *UAI*, pages 211–219. AUAI Press, 1998.

[8] Nicolas Meuleau, Leonid Peshkin, Kee eung Kim, and Leslie Pack Kaelbling. Learning finite-state controllers for partially observable environments. In *In Proceedings of the fifteenth conference on uncertainty in artificial intelligence*, pages 427–436. Morgan Kaufmann, 1999.

[9] Zhengzhu Feng and Eric Hansen. Approximate planning for factored pomdps. In *In Proceedings of the Sixth European Conference on Planning*. Springer, 2001.

[10] Pascal Poupart and Craig Boutilier. Bounded finite state controllers. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *NIPS*. MIT Press, Cambridge, MA, 2004.

[11] Shihao Ji, Ronald Parr, Hui Li, Xuejun Liao, and Lawrence Carin. Point-based policy iteration. In *AAAI'07: Proceedings of the 22nd national conference on Artificial intelligence*, pages 1243–1249. AAAI Press, 2007.

[12] Eric A. Hansen. Sparse stochastic finite-state controllers for POMDPs. In *UAI*, pages 256–263. AUAI Press, 2008.

[13] B. Bonet, H. Palacios, and H. Geffner. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *ICAPS, to appear*, 2009.

[14] Stéphane Ross, Joelle Pineau, Sébastien Paquet, and Brahim Chaibdraa. Online planning algorithms for pomdps. *J. Artif. Int. Res.*, 32(1):663–704, 2008.

[15] Steve Young, Milica Gasic, Simon Keizer, François Mairesse, Jost Schatzmann, Blaise Thomson, and Kai Yu. The hidden information state model: A practical framework for pomdp-based spoken dialogue management. *Computer Speech & Language*, 24(2):150–174, 2010.

[16] M. Sridharan, J. Wyatt, and R. Dearden. HiPPo: Hierarchical POMDPs for Planning Information Processing and Sensing Actions on a Robot. In *International Conference on Automated Planning and Scheduling (ICAPS)*, September 2008.

[17] Zeyn A. Saigol, Richard W. Dearden, Jeremy L. Wyatt, and Bramley J. Murton. Information-lookahead planning for AUV mapping. In *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI-09)*, 2009.

[18] Michael Kneebone and Richard Dearden. Navigation planning in probabilistic roadmaps with uncertainty. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press, september 2009.

[19] Eric A. Hansen and Shlomo Zilberstein. LAO * : A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1-2):35–62, 2001.

[20] Blai Bonet and Héctor Geffner. Solving pomdps: RTDP-bel vs. point-based algorithms. In *IJCAI*, page to appear, 2009.

[21] Olivier Buffet and Douglas Aberdeen. The factored policy-gradient planner. *Artificial Inteligence*, 173(5-6):722–747, 2009.

[22] Marc Hanheide, Nick Hawes, Jeremy Wyatt, Moritz Gobelbecker, Michael Brenner, Kristoffer Sjoo, Alper Aydemir, Patric Jensfelt, Hendrik Zender, and Geert-Jan M. Kruijff. A framework for goal generation and management. In *Proceedings of the AAAI Workshop on Goal-Directed Autonomy*, 2010.

[23] David K. Lewis. *Counterfactuals*. Harvard Univ. Press, Cambridge, MA, 1973.

[24] P. Gärdenfors. Belief revision and the Ramsey test for conditionals. *The Philosophical Review*, XCV(1):81–93, January 1986.

[25] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, April 1987.

[26] Thomas Eiter and Georg Gottlob. The complexity of logic-based abduction. *Jour. ACM*, 42(1):3–42, 1995.

[27] Adele Howe and Paul Cohen. Understanding planner behavior. *Artificial Intelligence, Special issue on Planning Systems.*, Vol. 76(1–2):125–166, 1995.

[28] M. Littman, J. Goldsmith, and M. Mundhenk. The computational complexity of probabilistic planning. *JAIR*, 9:1–36, 1998.

[29] Jussi Rintanen. Constructing conditional plans by a theorem-prover. *JAIR*, 10:323–352, 1999.

[30] José A. Ambros-Ingerson and Sam Steel. Integrating planning, execution and monitoring. In *Proc. AAAI-88*, pages 83–88, Saint Paul, MI, August 1988.

[31] K. Golden. Leap before you look: Information gathering in the puccini planner. In *Proc. AIPS-98*, pages 70–77, 1998.

[32] Oren Etzioni, Steve Hanks, Daniel Weld, Denise Draper, Neal Lesh, and Mike Williamson. An approach to planning with incomplete information. In *Proc. KR-92*, pages 115–125, 1992.

[33] Hector J. Levesque. What is planning in the presence of sensing? In *Proc. AAAI-96*, pages 1139–1146. MIT Press, July 1996.

[34] R. Petrick and F. Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In *Proc. AIPS-02*, 2002.

[35] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.

[36] Barbara J. Grosz and Sarit Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86, 1996.

[37] K. E. Lochbaum. A collaborative planning model of intentional structure. *Computational Linguistics*, 24:525–572, 1998.

[38] E. J. Sondik. *The Optimal Control of Partially Observable Markov Decision Processes*. PhD thesis, Stanford, California, 1971.

[39] Edward J. Sondik. The optimal control of partially observable markov processes over the infinite horizon: Discounted costs. *Operations Research*, 26(2):282–304, 1978.

[40] R. A. Howard. *Dynamic Probabilistic Systems*. John Wiley & Sons, New York., 1971.

[41] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[42] Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan. Learning without state-estimation in partially observable markovian decision processes. In *In Proceedings of the Eleventh International Conference on Machine Learning*, pages 284–292. Morgan Kaufmann Publishers Inc., 1994.

[43] Tommi Jaakkola, Satinder P. Singh, and Michael I. Jordan. Reinforcement learning algorithm for partially observable markov decision problems. In *NIPS*, pages 345–352. MIT Press, 1995.

[44] Jonathan Baxter and Peter L. Bartlett. Reinforcement learning in POMDPs via direct gradient ascent. In *ICML*, pages 41–48. Morgan Kaufmann Publishers Inc., 2000.

[45] Pascal Poupart. *Exploiting structure to efficiently solve large scale partially observable markov decision processes*. PhD thesis, University of Toronto, Toronto, Ont., Canada, Canada, 2005.

[46] Matthijs T. J. Spaan and Nikos Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195–220, 2005.

[47] Joelle Pineau, Geoffrey J. Gordon, and Sebastian Thrun. Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research*, 27:335–380, 2006.

[48] Josep M. Porta, Nikos Vlassis, Matthijs T. J. Spaan, and Pascal Poupart. Point-based value iteration for continuous POMDPs. *Journal of Machine Learning Research*, 7:2329–2367, 2006.

[49] Hctor Geffner and Blai Bonet. Solving large POMDPs using real time dynamic programming. In *AAAI Fall Symposium on POMDPs*, 1998.

[50] Trey Smith and Reid Simmons. Heuristic search value iteration for POMDPs. In *UAI*, pages 520–527, Arlington, Virginia, United States, 2004. AUAI Press.

[51] Trey Smith and Reid Simmons. Point-based POMDP algorithms: Improved analysis and implementation. In *UAI*, pages 542–55, Arlington, Virginia, 2005. AUAI Press.

[52] Darius Braziunas and Craig Boutilier. Stochastic local search for POMDP controllers. In *AAAI*, pages 690–696. AAAI Press, 2004.

[53] Xavier Boyen and Daphne Koller. Tractable inference for complex stochastic processes. In *UAI*, pages 33–42. AUAI Press, 1998.

[54] J. Bresina, R. Dearden, N. Meuleau, S. Ramkrishnan, D. Smith, and R. Washington. Planning under continuous time and resource uncertainty: A challenge for AI. In *Proc. of UAI-02*, pages 77–84. Morgan Kaufmann, 2002.

[55] Dan Weld, Denise Draper, and Steve Hanks. Probabilistic planning with information gathering and contingent execution. In *Proceedings of AIPS*, pages 31–36. AAAI Press, 1994.

[56] R. P. Bonasso, D. Kortenkamp, and T Whitney. Using a robot control architecture to automate space shuttle operations. In *Proceedings of IAAI*, 1997.

[57] D. K. Kortenkamp, R. P. Bonasso, and R. Murphy, editors. *AI-based Mobile Robots*. AAAI/MIT Press, 1997.

2 Annexes

2.1 Göbelbecker et al. "Coming up With Good Excuses: What to do When no Plan Can be Found" (ICAPS'10)

Bibliography Moritz Göbelbecker, Thomas Keller, Patrick Eyerich, Michael Brenner, and Bernhard Nebel. "Coming up With Good Excuses: What to do When no Plan Can be Found" In Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS 2010)

Abstract When using a planner-based agent architecture, many things can go wrong. First and foremost, an agent might fail to execute one of the planned actions for some reasons. Even more annoying, however, is a situation where the agent is incompetent, i.e., unable to come up with a plan. This might be due to the fact that there are principal reasons that prohibit a successful plan or simply because the task's description is incomplete or incorrect. In either case, an explanation for such a failure would be very helpful. We will address this problem and provide a formalization of coming up with excuses for not being able to find a plan. Based on that, we will present an algorithm that is able to find excuses and demonstrate that such excuses can be found in practical settings in reasonable time.

Relation to WP The phenomenon of a planner not being able to come up with a plan is of particular relevance to our robot applications, since the robot's limited perception and complex sensor fusion processes will often lead to incomplete information states and, consequentially, to the planner not being able to find a plan. Coming up with an "excuse", i.e., an explanation about what might be wrong, is a first step in making this failure transparent to a human user or triggering additional sensing that might provide the missing information. This will be demonstrated in the year 2 Dora system.

2.2 Benton et al. "G-value Plateaus: A Challenge for Planning" (ICAPS'10)

Bibliography J. Benton, Kartik Talamadupula, Patrick Eyerich, Robert Mattmüller, and Subbarao Kambhampati "G-value Plateaus: A Challenge for Planning" In Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS 2010)

Abstract Recent years have seen the development of several scalable planners, many of which follow the string of successes found in using heuristic, best-first search methods. While this provides positive reinforcement for

continuing work along these lines, fundamental problems arise when handling objectives whose value does not change with each search operation. An extreme case of this occurs when handling the objective of generating a temporal plan with short makespan. Typically used heuristic search methods assume strictly positive edge costs for their guarantees on completeness and optimality to hold, while the usual "fattening" and "advance time" steps of heuristic search algorithms for temporal planning have the potential for zero-cost edges, resulting in "g-value plateaus". In this paper we point out some underlying difficulties with using modern heuristic search methods for optimizing makespan and discuss how the presence of these problems contributes to the poor performance of makespan-optimizing heuristic search planners. To further illustrate this, we show empirical results on recent benchmarks using a planner made with makespan optimization in mind.

Relation to WP This is a "challenge paper" describing a problem typically faced by temporal planners (like our base planner Temporal Fast Downward).

2.3 Eyerich et al. "High-Quality Policies for the Canadian Traveler's Problem" (AAAI'10)

Bibliography Patrick Eyerich, Thomas Keller, and Malte Helmert "High-Quality Policies for the Canadian Traveler's Problem" In Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10).

Abstract We consider the stochastic variant of the Canadian Traveler's Problem, a path planning problem where adverse weather can cause some roads to be untraversable. The agent does not initially know which roads can be used. However, it knows a probability distribution for the weather, and it can observe the status of roads incident to its location. The objective is to find a policy with low expected travel cost. We introduce and compare several algorithms for the stochastic CTP. Unlike the optimistic approach most commonly considered in the literature, the new approaches we propose take uncertainty into account explicitly. We show that this property enables them to generate policies of much higher quality than the optimistic one, both theoretically and experimentally.

Relation to WP This paper is a first step towards extending our Continual Planning approach with a limited form of stochasticity, through Monte Carlo sampling. Here, we study this technique still within the context of a specific path planning problem, but will extend this to general planning tasks in the future.

2.4 Brenner "Creating Dynamic Story Plots with Continual Multiagent Planning" (AAAI'10)

Bibliography Michael Brenner "Creating Dynamic Story Plots with Continual Multiagent Planning" In Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10).

Abstract An AI system that is to create a story (autonomously or in interaction with human users) requires capabilities from many subfields of AI in order to create characters that themselves appear to act intelligently and believably in a coherent story world. Specifically, the system must be able to reason about the physical actions and verbal interactions of the characters as well as their perceptions of the world. Furthermore it must make the characters act believably–i.e. in a goal-directed yet emotionally plausible fashion. Finally, it must cope with (and embrace!) the dynamics of a multiagent environment where beliefs, sentiments, and goals may change during the course of a story and where plans are thwarted, adapted and dropped all the time. In this paper, we describe a representational and algorithmic framework for modelling such dynamic story worlds, Continual Multiagent Planning. It combines continual planning (i.e. an integrated approach to planning and execution) with a rich description language for modelling epistemic and affective states, desires and intentions, sensing and communication. Analysing story examples generated by our implemented system we show the benefits of such an integrated approach for dynamic plot generation.

Relation to WP The paper presents an extension of our Continual Planning approach to multiagent scenarios, such as the one our robot faces when interacting with humans. In such environments the robot must reason not only about its own actions, but also about what others will do and how the robot can make the engage in collaborative activity. Although the paper describes a different application in which virtual agents interact, the same principles apply to planning for human-robot interaction.

2.5 Wurm et al. "Coordinated Exploration with Marsupial Teams of Robots using Temporal Symbolic Planning" (IROS'10)

Bibliography Kai M. Wurm, Christian Dornhege, Patrick Eyerich, Cyrill Stachniss, Bernhard Nebel, Wolfram Burgard "Coordinated Exploration with Marsupial Teams of Robots using Temporal Symbolic Planning" In Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-10).

Abstract The problem of autonomously exploring an environment with a team of robots received considerable attention in the past. However, there are relatively few approaches to coordinate teams of robots that are able to deploy and retrieve other robots. Efficiently coordinating the exploration with such marsupial robots requires advanced planning mechanisms that are able to consider symbolic deployment and retrieval actions. In this paper, we propose a novel approach for coordinating the exploration with marsupial robot teams. Our method integrates a temporal symbolic planner that explicitly considers deployment and retrieval actions with a traditional utility-based assignment procedure. Our approach has been implemented and evaluated in several simulated environments and with varying team sizes. The results demonstrate that our proposed method is able to coordinate marsupial teams of robots to efficiently explore unknown environments.

Relation to WP This paper shows an application of our base planner, Temporal Fast Downward, to a multi-robot exploration scenario.

2.6 Robinson et al. "Partial Weighted MaxSAT for Optimal Planning" (PRICAI'10)

Bibliography N. Robinson, C. Gretton, D. Pham, and A. Sattar "Partial Weighted MaxSAT for Optimal Planning" In 11th Pacific Rim International Conference on Artificial Intelligence (PRICAI-2010).

Abstract We consider the problem of computing optimal plans for propositional planning problems with action costs. In the spirit of leveraging advances in general-purpose automated reasoning for that setting, we develop an approach that operates by solving a sequence of partial weighted MaxSAT problems, each of which corresponds to a step-bounded variant of the problem at hand. Our approach is the first SAT-based system in which a proof of cost optimality is obtained using a MaxSAT procedure. It is also the first system of this kind to incorporate an admissible planning heuristic. We perform a detailed empirical evaluation of our work using benchmarks from a number of International Planning Competitions.[5]

Relation to WP This paper demonstrates how to leveraged Boolean decision-procedures in order to solve complex optimisation problems in planning.

[5]An earlier version of this work was also published at a the ICAPS 2010 Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS'2010).

2.7 Dearden et al. "Robot Control Using a Switching Classical/Decision-Theoretic Planner" (in preparation for ICAPS 2011)

Bibliography R. Dearden, C. Gretton, M. Brenner, M. Göbelbacher "Robot Control Using a Switching Classical/Decision-Theoretic Planner" (unpublished draft).

Abstract Planning problems where the state of the world cannot be determined with certainty but must be inferred from observations are traditionally represented as partially observable Markov decision problems (POMDPs). However algorithms that operate on POMDPs are restricted to very small problems due to their computational cost. In this paper we describe an approach to plan in these domains by combining a classical planner and a POMDP planner. The approach builds plans as if the world was observable at execution time using the classical planner. These plans include assertions that state variables have particular values. When execution of the plan reaches one of these variables, the POMDP planner is called to generate a plan to determine the value of the variable. This plan can then be executed, the value determined, and the classical plan can then continue executing if the value determined was the one desired, or if the variable was found to have a different value, replanning occurs. We show how this approach can generate plans in domains that are much too large for POMDP planners to solve directly.

Relation to WP This draft paper describes the architecture of the switching planner and the process that is used to generate and execute a plan when the planner is called.

Coming up With Good Excuses: What to do When no Plan Can be Found

Moritz Göbelbecker and **Thomas Keller**
and **Patrick Eyerich** and **Michael Brenner** and **Bernhard Nebel**
University of Freiburg, Germany
{goebelbe, tkeller, eyerich, brenner, nebel}@informatik.uni-freiburg.de

Abstract

When using a planner-based agent architecture, many things can go wrong. First and foremost, an agent might fail to execute one of the planned actions for some reasons. Even more annoying, however, is a situation where the agent is *incompetent*, i.e., unable to come up with a plan. This might be due to the fact that there are principal reasons that prohibit a successful plan or simply because the task's description is incomplete or incorrect. In either case, an explanation for such a failure would be very helpful. We will address this problem and provide a formalization of *coming up with excuses* for not being able to find a plan. Based on that, we will present an algorithm that is able to find excuses and demonstrate that such excuses can be found in practical settings in reasonable time.

Introduction

Using a planner-based agent architecture has the advantage that the agent can cope with many different situations and goals in flexible ways. However, there is always the possibility that something goes wrong. For instance, the agent might fail to execute a planned action. This may happen because the environment has changed or because the agent is not perfect. In any case, recovering from such a situation by recognizing the failure followed by replanning is usually possible (Brenner and Nebel 2009).

Much more annoying than an execution failure is a failure to find a plan. Imagine a household robot located in the living room with a locked door to the kitchen that receives the order to tidy up the kitchen table but is unable to come up with a plan. Better than merely admitting it is *incompetent* would be if the robot could provide a good *excuse* – an explanation of *why* it was not able to find a plan. For example, the robot might recognize that if the kitchen door were unlocked it could achieve its goals.

In general, we will adopt the view that an excuse is a counterfactual statement (Lewis 1973) of the form that a small change of the planning task would permit the agent to find a plan. Such a statement is useful when debugging a domain description because it points to possible culprits that prevent finding a plan. Also in a regular setting a counterfactual explanation is useful because it provides a hint for where to start when trying to resolve the problem, e.g., by asking for help from a human or exploring the space of possible repair actions.

There are many ways to change a planning task so that it becomes possible to generate a plan. One may change

- the goal description,
- the initial state, or
- the set of planning operators.

Obviously, some changes are more reasonable than others. For example, weakening the goal formula is, of course, a possible way to go. We would then reduce the search for excuses to over-subscription planning (Smith 2004). However, simply ignoring goals would usually not be considered as an excuse or explanation.

On the other hand, changing the initial state appears to be reasonable, provided we do not make the trivial change of making goal atoms true. In the household robot example, changing the state of the door would lead to a solvable task and thus give the robot the possibility to actually realize the reasons of its inability to find a plan.

In some cases, it also makes sense to add new objects to the planning task, e.g., while the robot is still missing sensory information about part of its environment. Thus, we will consider changes to the object domain as potential changes of the task, too. Note that there are also situations in which removing objects is the only change to the initial state that may make the problem solvable. However, since these situations can almost always be captured by changing those objects' properties, we ignore this special case in the following.

Finally, changing the set of planning operators may indeed be a "better" way, e.g., if an operator to unlock the door is missing. However, because the number of potential changes to the set of operators exceeds the number of changes to the initial state by far, we will concentrate on the latter in the remainder of the paper, which also seems like the most intuitive type of explanation.

The rest of the paper is structured as follows. In the next section, we introduce the formalization of the planning framework we employ. After that we sketch a small motivating example. Based on that, we will formalize the notion of excuses and determine the computational complexity of finding excuses. On the practical side, we present a method

that is able to find good excuses, followed by a section that shows our method's feasibility by presenting empirical data on some IPC planning domains and on domains we have used in a robotic context. Finally, we discuss related work and conclude.

The Planning Framework

The planning framework we use in this paper is the ADL fragment of PDDL2.2 (Edelkamp and Hoffmann 2004) extended by multi-valued fluents as in the SAS$^+$ formalism (Bäckström and Nebel 1995) or functional STRIPS (Geffner 2000).[1] One reason for this extension is that modeling using multi-valued fluents is more intuitive. More importantly, changing fluent values when looking for excuses leads to more intuitive results than changing truth values of Boolean state variables, since we avoid states that violate implicit domain constraints. For example, if we represent the location of an object using a binary predicate $at(\cdot,\cdot)$, changing the truth value of a ground atom would often lead to having an object at two locations simultaneously or nowhere at all. The domain description does not tell us that, if we make a ground atom with the *at* predicate true, we have to make another ground atom with the identical first parameter false. By using multi-valued fluents instead, such implicit constraints are satisfied automatically.

Of course, our framework can be applied to any reasonable planning formalism, since it is simply a matter of convenience to have multi-valued fluents in the language. This being said, a **planning domain** is a tuple $\Delta = \langle \mathcal{T}, \mathcal{C}_\Delta, \mathcal{S}, \mathcal{O} \rangle$, where

- $\mathcal{T}$ are the **types**,
- $\mathcal{C}_\Delta$ is the set of **domain constant symbols**,
- $\mathcal{S}$ is the set of **fluent** and **predicate symbols** with associated arities and typing schemata, and
- $\mathcal{O}$ is the set of **planning operators** consisting of preconditions and effects.

A **planning task** is then a tuple $\Pi = \langle \Delta, \mathcal{C}_\Pi, s_0, s^* \rangle$, where

- Δ is a planning domain as defined above,
- $\mathcal{C}_\Pi$ is a set of **task-dependent constant symbols** disjoint from $\mathcal{C}_\Delta$,
- s_0 is the description of the **initial state**, and
- s^* is the **goal** specification.

The initial state is specified by providing a set s_0 of ground atoms, e.g., `(holding block1)` and ground fluent assignments, e.g., `(= (loc obj1) loc2)`. As usual, the description of the initial state is interpreted under the *closed world assumption*, i.e., any logical ground atom not mentioned in s_0 is assumed to be false and any fluent not mentioned is assumed to have an undefined value. In the following sections we assume that $\mathcal{S}$ contains only fluents and no predicates at all. More precisely, we will treat predicates as fluents with a domain of $\{\bot, \top\}$ and a default value of $\bot$ instead of unknown.

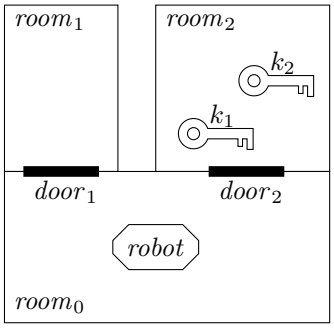
The goal specification is a closed first-order formula over logical atoms and fluent equalities. We say that a planning task is **solvable** iff there is a plan Ψ that transforms the state described by s_0 into a state that satisfies the goal specification s^*.

Sometimes we want to turn an initial state description into a (sub-)goal specification. Assuming that plan Ψ solves the task $\Pi = \langle \Delta, \mathcal{C}_\Pi, s_0, s^* \rangle$, by $nec(s_0, \Psi, s^*)$ we mean the formula that describes the setting of the fluents necessary for the correct execution of Ψ started in initial state s_0 leading to a state satisfying s^*. Note that $s_0 \models nec(s_0, \Psi, s^*)$ always holds.

Motivating Examples

The motivation for the work described in this paper mostly originated from the DESIRE project (Plöger et al. 2008), in which a household robot was developed that uses a domain-independent planning system. More often than not a sensor did not work the way it was supposed to, e.g., the vision component failed to detect an object on a table. If the user-given goal is only reachable by utilizing the missing object, the planning system naturally cannot find a plan. Obviously, thinking ahead of everything that might go wrong in a real-life environment is almost impossible, and if a domain-independent planning system is used, it is desirable to also realize flaws in a planning task with domain-independent methods. Furthermore, we wanted the robot to not only realize that something went wrong (which is not very hard to do after all), but it should also be able to tell the user what went wrong and why it couldn't execute a given command.

However, not only missing objects may cause problems for a planning system. Consider a simple planning task on the KEYS-domain, where a robot navigates between rooms through doors that can be unlocked by the robot if it has the respective key (see Fig. 1). The goal of such a task is to have the robot reach a certain room, which in this example is $room_1$.




```
(:objects
  room0 room1 room2 - room
  door1 door2 - door
  key1 key2 - key)

(:init
  (= (robot_pos) room0)
  (= (key_pos key1) room2)
  (= (key_pos key2) room1)
  (connects room0 room1 door1)
  (connects room1 room0 door1)
  (connects room0 room2 door2)
  (connects room2 room0 door2)
  (= (key_opens key1) door1)
  (= (key_opens key2) door2)
  (locked door1)
  (locked door2))

(:goal (= (robot_pos) room1))
```

Figure 1: Unsolvable task in the *Keys* domain with corresponding PDDL code.

Obviously there exists no plan for making the robot reach its goal. What, however, are the *reasons* for this task being

[1] Multi-valued fluents have been introduced to PDDL in version 3.1 under the name of "object fluents".

unsolvable? As we argued in the introduction, the answer to this question can be expressed as a counterfactual statement concerning the planning task. Of course, there are numerous ways to change the given problem such that a solution exists, the easiest one certainly being to already have the goal fulfilled in the initial state. An only slightly more complicated change would be to have the door to $room_1$ state) in which case the robot could directly move to its destination, or have the robot already carry the key to that door (changing the value of `(key_pos key1)` from `room2` to `robot`) or even a new one (adding an object of type key with the required properties), or simply have one of the keys in $room_0$ (e.g. `(= (key_pos key1) room0)`).

Having multiple possible excuses is, as in this case, rather the rule than the exception, and some of them are more reasonable than others. So, the following sections will answer two important questions. Given an unsolvable planning task, *What is a good excuse?* and *How to find a good excuse?*

Excuses

As spelled out above, for us an excuse is a change in the initial state (including the set of objects) with some important restrictions: We disallow the deletion of objects and changes to any fluents that could contribute to the goal. For example, we may not change the location of the robot if having the robot at a certain place is part of the goal.

A ground fluent f **contributes** to the goal if adding or deleting an assignment $f = x$ from a planning state can make the goal true. Formally, f contributes to s^* iff there exists a state s with $s \not\models s^*$ such that $s \cup \{f = x\} \models s^*$ for some value x.

Given an unsolvable planning task $\Pi = \langle \Delta, \mathcal{C}_\Pi, s_0, s^* \rangle$, an **excuse** is a tuple $\chi = \langle \mathcal{C}_\chi, s_\chi \rangle$ that implies the solvable **excuse task** $\Pi^\chi = \langle \Delta, \mathcal{C}_\chi, s_\chi, s^* \rangle$ such that $\mathcal{C}_\Pi \subseteq \mathcal{C}_\chi$ and if $(f = x) \in s_0 \triangle s_\chi$ (where $\triangle$ denotes the symmetric set difference) then f must not contribute to s^*.

The changed initial state s_χ is also called **excuse state**.

It should be noted that it is possible that no excuse exists, e.g., if there is no initial state such that a goal state is reachable. More precisely, there is no excuse iff the task is solvable or all changes to the initial state that respect the above mentioned restrictions do not lead to a solvable task.

Acceptable Excuses

If we have two excuses and one changes more initial facts than the other, it would not be an acceptable excuse, e.g., in our example above moving both keys to the room where the robot is would be an excuse. Relocating any one of them to that room would already suffice, though.

So, given two excuses $\chi = \langle \mathcal{C}_\chi, s_\chi \rangle$ and $\chi' = \langle \mathcal{C}_{\chi'}, s_{\chi'} \rangle$, we say that χ is at least as **acceptable** as χ', written $\chi \preceq \chi'$, iff $\mathcal{C}_\chi \subseteq \mathcal{C}_{\chi'}$ and $s_0 \triangle s_\chi \subseteq s_0 \triangle s_{\chi'}$. A minimal element under the ordering $\preceq$ is called an **acceptable excuse**.

Good Excuses

Given two acceptable excuses, it might nevertheless be the case that one of them subsumes the other if the changes in one excuse can be explained by the other one.

In the example from Fig. 1, one obvious excuse χ would lead to a task in which $door_1$ was unlocked so that the robot could enter $room_1$. This excuse, however, is unsatisfactory since the robot itself could unlock $door_1$ if its key was located in $room_0$ or if $door_2$ was unlocked. So any excuse χ' that contains one of these changes should subsume χ.

We can formalize this subsumption as follows: Let $\chi = \langle \mathcal{C}_\chi, s_\chi \rangle$ be an acceptable excuse to a planning task $\Pi = \langle \Delta, \mathcal{C}_\Pi, s_0, s^* \rangle$ with the plan Ψ solving Π^χ. Another acceptable excuse $\chi' = \langle \mathcal{C}_{\chi'}, s_{\chi'} \rangle$ to Π is called **at least as good as** χ, in symbols $\chi' \sqsubseteq \chi$, if χ' is an acceptable excuse also to $\langle \Delta, \mathcal{C}_\Pi, s_0, \mathit{nec}(s_{\chi'}, \Psi, s^*) \rangle$. We call χ' better than χ, in symbols $\chi' \sqsubset \chi$, iff $\chi' \sqsubseteq \chi$ and $\chi \not\sqsubseteq \chi'$.

In general, good excuses would be expected to consist of changes to so-called *static facts*, facts that cannot be changed by the planner and thus cannot be further regressed from, as captured by the above definition. In our example this could be a new key – with certain properties – and perhaps some additional unlocked doors between rooms.

However, there is also the possibility that there are cyclic dependencies as in the children's song *There's a Hole in the Bucket*. In our example, one excuse would be χ, where the door to $room_2$ is unlocked. In a second one, χ', the robot carries key k_2. Obviously, $\chi \sqsubseteq \chi'$ and $\chi' \sqsubseteq \chi$ hold and thus χ and χ' form a cycle in which all excuses are equally good.

In cases with cyclic dependencies, it is still possible to find even "better" excuses by introducing additional objects, e.g., a new door or a new key in our example. However, cyclic excuses as above, consisting of χ and χ', appear to be at least as intuitive as excuses with additional objects. For these reasons, we define a **good** excuse χ as one such that there either is no better excuse or there exists a different excuse χ' such that $\chi \sqsubseteq \chi'$ and $\chi' \sqsubseteq \chi$.

Perfect Excuses

Of course, there can be many good excuses for a task, and one may want to distinguish between them. A natural way to do so is to introduce a cost function that describes the cost to transform one state into another. Of course, such a cost function is just an estimate because the planner has no way to transform the initial state into the excuse state.

Here, we will use a cost function $c(\cdot)$, which should respect the above mentioned acceptability ordering $\preceq$ as a minimal requirement. So, if $\chi' \preceq \chi$, we require that $c(\chi') \leq c(\chi)$. As a simplifying assumption, we will only consider additive cost functions. So, all ground fluents have predefined costs, and the cost of an excuse is simply the sum over the costs of all facts in the symmetric difference between initial and excuse state. Good excuses with minimal costs are called **perfect excuses**.

Computational Complexity

In the following, we consider ordinary propositional and DATALOG planning, for which the problem of deciding plan existence – the PLANEX problem – is PSPACE- and EXPSPACE-complete, respectively (Erol, Nau, and Subrahmanian 1995). In the context of finding excuses, we will mainly consider the following problem for acceptable, good, and perfect excuses:

- EXCUSE-EXIST: Does there exist any excuse at all?

In its unrestricted form, this problems is undecidable for DATALOG planning.

Theorem 1 *EXCUSE-EXIST is undecidable for* DATALOG *planning.*

Proof Sketch. The main idea is to allow an excuse to introduce an unlimited number of new objects, which are arranged as tape cells of a Turing machine. That these tape cells are empty and have the right structure could be verified by an operator that must be executed in the beginning. After that a Turing machine could be simulated using ideas as in Bylander's proof (Bylander 1994). This implies that the Halting problem on the empty tape can be reduced to EXCUSE-EXIST, which means that the latter is undecidable. ∎

However, an excuse with an unlimited number of new objects is, of course, also not very intuitive. For these reasons, we will only consider BOUNDED-EXCUSE-EXIST, where only a polynomial number of new objects is permitted. As it turns out, this version of the problem is not more difficult than planning.

Lemma 2 *There is a polynomial Turing reduction from PLANEX to BOUNDED-EXCUSE-EXIST for acceptable, good, or perfect excuses.*

Proof Sketch. Given a planning task $\Pi = \langle \Delta, \mathcal{C}_\Pi, s_0, s^* \rangle$ with planning domain $\Delta = \langle \mathcal{T}, \mathcal{C}_\Delta, \mathcal{S}, \mathcal{O} \rangle$, construct two new tasks by extending the set of predicates in $\mathcal{S}$ by a fresh ground atom a leading to $\mathcal{S}'$. In addition, this atom is added to all preconditions in the set of operators resulting in $\mathcal{O}'$. Now we generate:

$$\begin{aligned} \Pi' &= \langle \langle \mathcal{T}, \mathcal{C}_\Delta, \mathcal{S}', \mathcal{O}' \rangle, \mathcal{C}_\Pi, s_0, s^* \rangle \\ \Pi'' &= \langle \langle \mathcal{T}, \mathcal{C}_\Delta, \mathcal{S}', \mathcal{O}' \rangle, \mathcal{C}_\Pi, s_0 \cup \{a\}, s^* \rangle \end{aligned}$$

Obviously, Π is solvable iff there exists an excuse for Π' and there is no excuse for Π''. ∎

It is also possible to reduce the problems the other way around, provided the planning problems are in a deterministic space class.

Lemma 3 *The BOUNDED-EXCUSE-EXIST problem can be Turing reduced to the PLANEX problem – provided PLANEX is complete for a space class that includes PSPACE.*

Proof Sketch. By Savitch's theorem (1980), we know that NSPACE$(f(n)) \subseteq$ DSPACE$((f(n))^2)$, i.e., that all deterministic space classes including PSPACE are equivalent to their non-deterministic counterparts. This is the main reason why finding excuses is not harder than planning.

Let us assume that the plan existence problem for our formalism is XSPACE-complete. Given a planning task $\Pi = \langle \Delta, \mathcal{C}_\Pi, s_0, s^* \rangle$, the following algorithm will determine whether there is an excuse:

1. If Π is solvable, return "no".
2. Guess a $\chi = \langle \mathcal{C}_\chi, s_\chi \rangle$ and verify the following:
   a) $\mathcal{C}_\Pi \subseteq \mathcal{C}_\chi$;
   b) $\Pi^\chi = \langle \Delta, \mathcal{C}_\chi, s_\chi, s^* \rangle$ is solvable;

This non-deterministic algorithm obviously needs only XSPACE using an XSPACE-oracle for the PLANEX problem. Since the existence of an excuse implies that there is a perfect excuse (there are only finitely many different possible initial states), the algorithm works for all types of excuses. ∎

From the two lemmas, it follows immediately that the EXCUSE-EXIST problem and the PLANEX problem have the same computational complexity.

Theorem 4 *The BOUNDED-EXCUSE-EXIST problem is complete for the same complexity class as the PLANEX problem for all planning formalisms having a PLANEX problem that is complete for a space class containing PSPACE.*

Using similar arguments, it can be shown that we can compute which literals in the initial state can be relevant or are necessary for an excuse. By guessing and verifying using PLANEX-oracles, these problems can be solved and are therefore in the same space class as the PLANEX problems, provided they are complete for a space class including PSPACE.

Candidates for Good Excuses

The range of changes that may occur in acceptable excuses is quite broad: The only excuses forbidden are those that immediately contribute to the goal. We could try to find acceptable excuses and apply goal regression until we find a good excuse, but this would be highly suboptimal, because it might require a lot of goal regression steps. Therefore, we first want to explore if there are any constraints (on fluent or predicate symbols, source or target values) that must be satisfied in any *good* excuse.

In order to analyze the relations between fluent symbols, we apply the notion of *causal graphs* and *domain transition graphs* (Helmert 2006) to the abstract domain description.

The **causal graph** CG_Δ of a planning domain $\Delta = \langle \mathcal{T}, \mathcal{C}_\Delta, \mathcal{S}, \mathcal{O} \rangle$ is a directed graph $(\mathcal{S}, A)$ with an arc $(u, v) \in A$ if there exists an operator $o \in \mathcal{O}$ so that $u \in \mathbf{pre}(o)$ and $v \in \mathbf{eff}(o)$ or both u and v occur in $\mathbf{eff}(o)$. If $u = v$ then (u, v) is in A iff the fluents in the precondition and effect can refer to distinct instances of the fluent.

The causal graph captures the dependencies of fluents on each other; to analyze the ways the values of one fluent can change, we build its domain transition graph. In contrast to the usual definition of domain transition graphs (which is based on grounded planning tasks), the domain of a fluent f can consist of constants as well as free variables. This fact needs to be taken into account when making statements about the domain transition graph (e.g., the reachability of a variable of type t implies the reachability of all variables of subtypes of t). For the sake of clarity, we will largely gloss over this distinction here and treat the graph like its grounded counterpart.

If $dom(f)$ is the domain of a fluent symbol $f \in \mathcal{S}$, its **domain transition graph** $\mathcal{G}_f$ is a labeled directed graph

$(dom(f), E)$ with an arc $(u, v) \in E$ iff there is an operator $o \in \mathcal{O}$ so that $f = u$ is contained in $\mathbf{pre}(o)$ and $f = v \in \mathbf{eff}(o)$. The label consists of the preconditions of o minus the precondition $f = u$. An arc from the unknown symbol, $(\bot, v)$, exists if f does not occur in the precondition. We also call such an arc $(u, v) \in \mathcal{G}_f$ a **transition** of f from u to v and the label of (u, v) its precondition.

For example, the domain transition graph of the `robot_pos` fluent has one vertex consisting of a variable of type `room` and one edge $(\texttt{room}, \texttt{room})$ with the label of $\texttt{connected}(\mathit{room_1}, \mathit{room_2}, \mathit{door}) \wedge \texttt{open}(\mathit{door})$.

In order to constrain the set of possible excuses, we restrict candidates to those fluents and values that are relevant for achieving the goal. The **relevant domain**, $dom_{rel}(f)$, of a fluent f is defined by the following two conditions and can be calculated using a fixpoint iteration: If $f = v$ contributes to the goal, then $v \in dom_{rel}(f)$. Furthermore, for each fluent f' on which f depends, $dom_{rel}(f')$ contains the subset of $dom(f')$ which is (potentially) required to reach any element of $dom_{rel}(f)$.

A **static change** is a change for which there is no path in the domain-transition graph even if all labels are ignored. Obviously all changes to *static variables* are static, but the converse is not always true. For example, in most planning domains, if in a planning task a non-static fluent f is undefined, setting f to some value x would be a static change.

In the following, we show that in some cases it is sufficient to consider static changes as candidates for excuses in order to find all good excuses. To describe these cases, we define two criteria, mutex-freeness and strong connectedness, that must hold for static and non-static fluents, respectively.

We call a fluent f **mutex-free** iff changing the value of an instance of f in order to enable a particular transition of a fluent f' that depends on f does not prevent any other transition. Roughly speaking, excuses involving f' are not good, because they can always be regressed to the dependencies f without breaking anything else. Two special cases of mutex-free fluents are noteworthy, as they occur frequently and can easily be found by analyzing the domain description: If a fluent f is *single-valued*, i.e. there are no two operators which depend on different values for f, it is obviously mutex-free. A less obvious case is free variables. Let o be an operator that changes the fluent $f(p_1, \ldots p_n)$ from p_v to p'_v. A precondition $f'(q_1, \ldots q_n) = q_v$ of o has free variables iff there is at least one variable in $q_1, \ldots, q_n$ that doesn't occur in $\{p_1, \ldots p_n, p_v, p'_v\}$. Here the mutex-freeness is provided because we can freely add new objects to the planning task and thus get new grounded fluents that cannot interfere with any existing fluents.

For example, consider the `unlock` operator in the KEYS-domain. Its precondition includes $\texttt{key\_pos}(\mathit{key}) = \texttt{robot} \wedge \texttt{key\_opens}(\mathit{key}) = \mathit{door}$. Here key is a free variable, so it is always possible to satisfy this part of the precondition by introducing a new key object and setting its position to the robot and its `key_opens` property to the door we want to open. As we do not have to modify an existing key, all actions that were previously applicable remain so.

The second criterion is the connectedness of the domain transition graph. We call a fluent f **strongly connected** iff the subgraph of $\mathcal{G}_f$ induced by the relevant domain of f is strongly connected. This means that once f has a value in $dom_{rel}(f)$ any value that may be relevant for achieving the goal can be reached in principle. In practice, most fluents have this property because any operator that changes a fluent from one free variable to another connects all elements of that variable's type.

Theorem 5 *Let Δ be a domain with an acyclic causal graph where all non-static fluents are strongly connected and all static fluents are mutex-free.*

Then any good excuse will only contain static changes.

Proof. First note that a cycle free causal graph implies that there are no co-occurring effects, as those would cause a cycle between their fluents.

If an excuse $\chi = \langle \mathcal{C}_\chi, s_\chi \rangle$ for $\Pi = \langle \Delta, \mathcal{C}_\Pi, s_0, s^* \rangle$ is an excuse that contains non-static changes, then we will construct an excuse $\chi' = \langle \mathcal{C}_{\chi'}, s_{\chi'} \rangle \sqsubset \chi$ containing only static facts which can explain χ. As all static fluents are mutex-free, no changes made to the initial state $s_{\chi'}$ to fulfill static preconditions can conflict with changes already made to s_χ, so we can choose the static changes in χ' to be those that make all (relevant) static preconditions true.

Let f, v, v' be a non-static change, i.e., $f = v \in s_0$ and $f = v' \in s_\chi$. This means that there exists a path from v to v' in $\mathcal{G}_f$. If all preconditions along this path are static, we are done as all static preconditions are satisfied in $s_{\chi'}$. If there are non-static preconditions along the path from v to v', we can apply this concept recursively to the fluents of those preconditions. As there are no co-occurring effects and the relevant part of each non-static fluent's domain transition graph is strongly connected, we can achieve all preconditions for each action and restore the original state later. ∎

We can easily extend this result to domains with a cyclic causal graph:

Theorem 6 *Let Δ be a domain where all non-static fluents are strongly connected, all static fluents are mutex-free and each cycle in the domain's causal graph contains at least one mutex-free fluent.*

Then any good excuse will only contain static changes or changes that involve a fluent on a cycle.

Proof. We can reduce this case to the non-cyclic case, by removing all effects that change the fluents f fulfilling the mutex-free condition, thus making them static. Let us call this modified domain Δ'.

Let χ be an excuse with non-static changes. Because Δ' contains only a subset of operators of Δ, any non-static change that can further be explained in Δ' can also be explained in Δ. So there exists an $\chi' \sqsubset \chi$, which means that χ cannot be a good excuse unless the changed fluent lies on a cycle so that $\chi \sqsubset \chi'$ may hold, too. ∎

While these conditions may not apply to all common planning domains as a whole, they usually apply to a large enough set of the fluents so that limiting the search to static and cyclic excuses speeds up the search for excuses significantly without a big trade-off in optimality.

Finding Excuses Using a Cost-Optimal Planner

We use the results from the previous section to transform the problem of finding excuses into a planning problem by adding operators that change those fluents that are candidates for good excuses. If we make sure that those **change operators** can only occur at the start of a plan, we get an excuse state s_χ by applying them to s_0.

Given an (unsolvable) planning task $\Pi = \langle \Delta, \mathcal{C}_\Pi, s_0, s^* \rangle$, we create a transformed task with action costs $\Pi' = \langle \Delta', \mathcal{C}_{\Pi'}, {s_0}', s^* \rangle$ as follows.

We recursively generate the relevant domain for each fluent symbol $f \in \mathcal{S}$ by traversing the causal graph, starting with the goal symbols. During this process, we also identify cyclic dependencies. Then we check $\mathcal{G}_f$ for reachability, adding all elements of $dom(f)$ from which $dom_{rel}(f)$ is **not** reachable to $changes(f)$. If f is involved in a cyclic dependency we also add $dom_{rel}(f)$ to $changes(f)$.

To prevent further changes to the planning state after the first execution of a regular action, we add the predicate $started$ to $\mathcal{S}'$ and as a positive literal to the effects of all operators $o \in \mathcal{O}$.

For every fluent f (with arity n) and $v \in changes(f)$ we introduce a new operator set_v^f as follows:

$$\mathbf{pre}(set_v^f) = \neg started \ \wedge f(p_1 \dots p_n) = v \bigwedge_{i=1}^{n} \neg unused(p_i)$$

$$\mathbf{eff}(set_v^f) = \{f(p_1 \dots p_n) = p_{n+1}\}$$

To add a new object of type t to the initial state, we add a number[2] of **spare objects** $sp_1^t \dots sp_n^t$ to $\mathcal{C}_{\Pi'}$. For each of these objects sp_i^t, the initial state ${s_0}'$ contains the facts $unused(sp_i^t)$. We then add the operator $add^t(p)$ with $\mathbf{pre}(add^t) = unused(p) \wedge \neg started$ and $\mathbf{eff}(add^t) = \{\neg unused(p)\}$.

To prevent the use of objects that have not been activated yet we add $\neg unused(p_i)$ to each operator $o \in \mathcal{O}$ for each parameter p_i to $pre(o)$ if $pre(o)$ does not contain a fluent or positive literal with p_i as parameter.

Due to the use of the $started$ predicate, any plan Ψ can be partitioned into the actions before $started$ was set (those that change the initial state) and those after. We call the subplans Ψ_{s_0} and Ψ_Π, respectively.

As a final step we need to set the costs of the change actions. In this implementation we assume an additive cost function that assigns non-zero costs to each change and does not distinguish between different instances of a fluent, so $c(f)$ are the costs associated with changing the fluent f and $c(t)$ the costs of adding an object of type t. We set $c(set_v^f) = \alpha c(f)$ and $c(add^t) = \alpha c(t)$ with α being a scaling constant. We need to make sure that the costs of the change actions always dominate the total plan's costs as otherwise worse excuses might be found if they cause Ψ_Π to be shorter. We can achieve this by setting α to an appropriate upper bound of the plan length in the original problem Π.

From the resulting plan Ψ we can easily construct an excuse $\chi = \langle \mathcal{C}_\Psi, s_\Psi \rangle$ with $\mathcal{C}_\Psi = \mathcal{C}_\Pi \cup \{c : add^t(c) \in \Psi\}$ and s_Ψ being the state resulting from the execution of Ψ_{s_0} restricted to the fluents defined in the original Problem Π.

Theorem 7 *Let Π be a planning task, Ψ an optimal solution to the transformed task Π', and $\chi = \langle \mathcal{C}_\Psi, s_\Psi \rangle$ the excuse constructed from Ψ. Then χ is an acceptable excuse to Π.*

Proof. Ψ_Π only contains operators in Δ and constants from $\mathcal{C}_\chi$. Obviously Ψ_Π also reaches the goal from s_Ψ. So Π^χ is solvable and χ thus an excuse. To show that χ is acceptable, we need to show that no excuse with a subset of changes exists. If such an excuse χ' existed it could be reached by applying change operators (as the changes in χ' are a subset of those in χ). Then a plan Ψ' would exist with $c(\Psi'_{s_0}) < c(\Psi_{s_0})$ and, as the cost of Ψ_{s_0} always dominates the cost of Ψ_Π, $c(\Psi') < c(\Psi)$. This contradicts that Ψ is optimal, so χ must be acceptable. ∎

Theorem 8 *Let Π be a planning task, Ψ an optimal solution to the transformed task Π', and $\chi = \langle \mathcal{C}_\Psi, s_\Psi \rangle$ the excuse constructed from Ψ. If χ changes only static facts, it is a perfect excuse.*

Proof. As χ contains only static facts, it must be a good excuse. From the definition of the cost function it follows that $c(\chi) = \alpha c(\Psi_{s_0})$, so existence of an excuse χ' with $c(\chi') < c(\chi)$ would imply, as in the previous proof, the existence of a plan Ψ' with $c(\Psi') < c(\Psi)$, contradicting the assumption that Ψ is optimal. ∎

Cyclic Excuses

Solving the optimal planning problem will not necessarily give us a good excuse (unless the problem's causal graph is non-cyclic, of course). So if we get an excuse that changes a non-static fact, we perform a *goal regression* as described earlier. We terminate this regression when all new excuses have already been encountered in previous iterations or no excuse can be found anymore. In the former case we select the excuse with the lowest cost from the cycle, in the latter case we need to choose the last found excuse.

Note though, that this procedure will not necessarily find excuses with globally optimal costs: As there is no guarantee that $\chi' \sqsubset \chi$ also implies $c(\chi') \leq c(\chi)$ the goal regression might find excuses that have higher costs than a good excuse that might be found from the initial task Π.

Experiments

To test our implementation's quality we converted selected planning tasks of the IPC domains LOGISTICS (IPC'00), ROVERS and STORAGE (both IPC'06) to use object fluents, so that our algorithm could work directly on each problem's SAS^+ representation. In order to give our program a reason to actually search for excuses, it was necessary to create flaws in each problem's description that made it unsolvable. For each problem file, we modified the initial state by randomly deleting any number of valid fluents and predicates,

[2] As shown in the complexity discussion, the number of new objects might be unreasonably high. In some cases this number can be restricted further but this has been left out for space reasons. In practice we cap the number of spares per type with a small constant.

	sat 0	**opt** 0	**sat** 1	**opt** 1	**sat** 2	**opt** 2	**sat** 3	**opt** 3	**sat** 4	**opt** 4
logistics-04	0.78s	1.43s	0.69s (0.5)	0.94s (0.5)	0.71s (1.5)	1.02s (1.5)	0.53s (1.0)	0.57s (1.0)	0.52s (2.5)	1.29s (2.5)
logistics-06	0.75s	9.81s	0.74s (1.5)	28.12s (1.5)	0.65s (2.5)	101.47s (2.5)	0.65s (3.0)	55.05s (2.5)	0.62s (3.5)	43.57s (3.5)
logistics-08	1.27s	76.80s	1.27s (1.0)	276.99s (1.0)	1.17s (1.0)	46.47s (1.0)	1.08s (5.5)	1176.49s (3.5)	0.96s (5.5)	1759.87s (4.5)
logistics-10	2.62s	—	2.24s (2.0)	—	2.36s (5.5)	—	2.25s (4.0)	—	1.29s (5.5)	—
logistics-12	2.58s	—	2.66s (2.0)	—	2.66s (4.5)	—	2.28s (5.0)	—	1.89s (6.5)	—
logistics-14	4.73s	—	4.78s (2.5)	—	4.24s (6.0)	—	3.70s (7.5)	—	2.71s (6.0)	—
rovers-01	3.04s	3.61s	3.09s (0.5)	5.72s (0.5)	3.17s (1.5)	8.17s (1.5)	2.79s (5.5)	—	2.90s (7.5)	—
rovers-02	3.25s	3.79s	3.24s (0.5)	4.45s (0.5)	3.31s (2.5)	21.48s (2.5)	3.23s (3.0)	62.36s (3.0)	2.87s (6.5)	—
rovers-03	4.15s	5.53s	4.11s (0.5)	7.90s (0.5)	3.55s (2.5)	112.43s (2.5)	4.04s (5.5)	—	3.67s (6.5)	—
rovers-04	5.01s	6.53s	4.94s (1.0)	8.97s (0.5)	68.60s (5.0)	22.01s (2.0)	3.21s (6.0)	—	9.45s (12.0)	—
rovers-05	5.29s	—	6.23s (2.0)	925.61s (2.0)	7.25s (4.0)	—	5.82s (5.0)	790.57s (5.0)	6.32s (8.0)	—
storage-01	1.77s	1.83s	2.01s (0.5)	2.31s (0.5)	1.71s (3.0)	2.11s (2.0)	1.84s (5.0)	24.81s (4.0)	1.82s (4.5)	11.12s (3.5)
storage-05	11.14s	15.66s	10.85s (0.5)	37.09s (0.5)	8.25s (4.0)	53.38s (4.0)	10.25s (6.0)	—	31.70s (6.0)	—
storage-08	30.46s	101.32s	35.59s (1.5)	—	774.17s (5.5)	—	765.32s (7.5)	—	110.31s (8.5)	—
storage-10	88.07s	214.10s	62.93s (1.0)	—	64.56s (2.0)	—	423.71s (3.0)	—	257.10s (4.0)	—
storage-12	131.36s	—	—	—	—	—	—	—	—	—
storage-15	1383.65s	—	—	—	—	—	—	—	—	—

Table 1: Results for finding excuses on some IPC domains. All experiments were conducted on a 2.66 GHz Intel Xeon processor with a 30 minutes timeout and a 2 GB memory limit. We used two setting for the underlying Fast Downward Planner: **sat** is Weighted A* with the enhanced-additive heuristic and a weight of 5, **opt** is A* with the admissible LM Cut Heuristic. For each problem instance there are five versions: the original (solvable) version is referred to as 0 while versions 1 to 4 are generated according to the deletions described in the Experiments section. We used an uniform cost measure with the exception that assigning a value to a previously undefined fluent costs 0.5. Runtime results are in seconds; the excuses costs are shown in parentheses.

or by completely deleting one or more objects necessary to reach the goal in every possible plan (the latter includes the deletion of all fluents and predicates containing the deleted object as a parameter). For instance, in the LOGISTICS domain, we either deleted one or more *city-of* fluents, or all trucks located in the same city, or all airplanes present in the problem.

In order to not only test on problems that vary in the difficulty to find a plan, but also the difficulty to find excuses, we repeated this process four times, each repetition taking the problem gained in the iteration before as the starting point. This lead to four versions of each planning task, each one missing more initial facts compared to the original task than the one before.

Our implementation is based on the Fast Downward planning system (Helmert 2006), using a Weighted A^* search with an extended context-enhanced additive heuristic that takes action costs into account. Depicted are runtimes and the cost of the excuse found. Because this heuristic is not admissible, the results are not guaranteed to be optimal, so we additionally ran tests using A^* and the (admissible) landmark cut heuristics (Helmert and Domshlak 2009).

To judge the quality of the excuses produced we used a uniform cost measure, with one exception: The cost of the assignment of a concrete value to a previously undefined fluent is set to be 0.5. This kind of definition captures our definition of acceptable excuses via the symmetric set difference and also appears to be natural: Assigning a value to an undefined fluent should be of lower cost than changing a value that was given in the original task. Note that switching a fluent's value actually has a cost of 1.0.

As the results in Table 1 show, the time for finding excuses increases significantly in the larger problems. The principal reason for that is that previously static predicates like `connected` in the STORAGE domain have become non-static due to the introduction of change operators. This leads both to a much larger planning state (as they cannot be compiled away anymore), as well as a much larger amount of applicable ground actions. This effect can be seen in the first two columns which show the planning times on the unmodified problems (but with the added change operators).

As expected, optimal search was able to find excuses for fewer problems than satisficing search. Satisficing search came up with excuses for most problems in a few seconds with the exception of the storage domain, due to the many static predicates. The costs of the excuses found were sometimes worse than those found by optimal search, but usually not by a huge amount. If better excuses are desired, additional tests showed that using smaller weights for the Weighted A^* are a reasonable compromise.

It is interesting to note that for the satisficing planner the number of changes needed to get a solvable task has little impact on the planning time. The optimal search, on the other hand, usually takes much longer for the more flawed problems. A possible explanation for this behavior is that the number of possible acceptable excuses grows drastically the more facts we remove from the problem. This makes finding *some* excuse little harder, but greatly increases the difficulty of finding an *optimal* excuse.

While most of the excuses described in this paper can be found in the problems we created this way, it is very unlikely that a problem is contained that is unsolvable due to a cyclic excuse[3]. The aforementioned KEYS-domain on the other hand is predestined to easily create problems that are unsolvable because of some cyclic excuse. So for our second experiment, we designed problems on that domain with an increasing number of rooms n connected so that they form a cycle: for each room k, $k \neq n$, there is a locked door k leading to room $k+1$, and an additional, unlocked one between rooms n and 0 (each connection being valid only in the described direction). For each door k there is a key k which is placed in room k, with the exception of key 0 which is placed in room n and the key to the already unlocked door n which doesn't exist. Obviously a good excuse for every n remains the same: If the robot held the key to door 0 in the

[3]This is only possible if that cyclic excuse was already part of the task, but didn't cause a problem because there existed another, after the deletion nonexistent way to the goal.

rooms	sat	opt	rooms	sat	opt
3	0.91s (1)	0.97s (1)	10	19.20s (2)	368.09s (1)
4	1.2s (1)	1.72s (1)	11	57.39s (2)	849.69s (1)
5	1.75s (1)	4.23s (1)	12	72.65s (2)	1175.23s (1)
6	2.19s (2)	10.69s (1)	13	84.45s (2)	—
7	4.24s (2)	27.01s (1)	14	215.05s (2)	—
8	6.03s (2)	65.15s (1)	15	260.39s (2)	—
9	14.22s (2)	158.28s (1)	16	821.82s (2)	—

Table 2: Results for finding excuses on the KEYS domain. We used the same settings as in the experiments for Table 1, except that the weight in the satisficing run was 1. The rows labeled **rooms** give the number of rooms or the size of the cycle minus 1.

initial state, or if that door was unlocked, the task would easily be solvable. The number of necessary regression steps to find that excuse grows with n, though, which is why KEYS is very well suited to test the performance of the finding cyclic excuses part of our implementation.

As can be seen in the results in Table 2, the planning time both scales well with the size of the cycle and is reasonable for practical purposes.

Related Work

We are not aware of any work in the area of AI planning that addresses the problem of explaining why a goal cannot be reached. However, as mentioned already, there is some overlap with abduction (a term introduced by the philosopher Peirce), counterfactual reasoning (Lewis 1973), belief revision (Gärdenfors 1986), and consistency-based diagnosis (Reiter 1987). All these frameworks deal with identifying a set of propositions or beliefs that either lead to inconsistencies or permit to deduce an observation. There are parallels to our notions of acceptable, good and perfect approaches in these fields (Eiter and Gottlob 1995) – for non-cyclic excuses. The main difference to the logic-based frameworks is that in our case there is no propositional or first-order background theory. Instead, we have a set of operators that allows us to transform states. This difference might be an explanation why cyclic excuses are something that appear to be relevant in our context, but have not been considered as interesting in a purely logic-based context.

Conclusion

In this paper we have investigated situations in which a planner-based agent is incompetent to find a solution for a given planning task. We have defined what an *excuse* in such a situation might look like, and what characteristics such an excuse must fulfill to be accounted as *acceptable*, *good* or even *perfect*. Our main theoretical contribution is a thorough formal analysis of the resulting problem along with the description of a concrete method for finding excuses utilizing existing classical planning systems. On the practical side, we have implemented this method resulting in a system that is capable of finding even complicated excuses in reasonable time which is very helpful both for debugging purposes and in a regular setting like planner-based robot control.

As future work, we intend to extend our implementation to more expressive planning formalisms dealing with time and resources.

Acknowledgements

This research was partially supported by DFG as part of the collaborative research center SFB/TR-8 Spatial Cognition Project R7, the German Federal Ministry of Education and Research (BMBF) under grant no. 01IME01-ALU (DESIRE) and by the EU as part of the Integrated Project CogX (FP7-ICT-2xo15181-CogX).

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS^+ planning. *Comp. Intell.* 11(4):625–655.

Brenner, M., and Nebel, B. 2009. Continual planning and acting in dynamic multiagent environments. *JAAMAS* 19(3):297–331.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *AIJ* 69(1–2):165–204.

Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The language for the classical part of the 4th international planning competition. Technical Report 195, Univ. Freiburg, Institut für Informatik, Freiburg, Germany.

Eiter, T., and Gottlob, G. 1995. The complexity of logic-based abduction. *Jour. ACM* 42(1):3–42.

Erol, K.; Nau, D. S.; and Subrahmanian, V. S. 1995. Complexity, decidability and undecidability results for domain-independent planning. *AIJ* 76(1–2):75–88.

Gärdenfors, P. 1986. Belief revision and the Ramsey test for conditionals. *The Philosophical Review* XCV(1):81–93.

Geffner, H. 2000. Functional STRIPS: a more flexible language for planning and problem solving. In Minker, J., ed., *Logic-Based Artificial Intelligence*. Dordrecht, Holland: Kluwer.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *ICAPS 2009*, 162–169.

Helmert, M. 2006. The fast downward planning system. *JAIR* 26:191–246.

Lewis, D. K. 1973. *Counterfactuals*. Cambridge, MA: Harvard Univ. Press.

Plöger, P.-G.; Pervölz, K.; Mies, C.; Eyerich, P.; Brenner, M.; and Nebel, B. 2008. The DESIRE service robotics initiative. *KI* 4:29–32.

Reiter, R. 1987. A theory of diagnosis from first principles. *AIJ* 32(1):57–95.

Savitch, W. J. 1980. Relations between nondeterministic and deterministic tape complexity. *Journal of Computer and System Sciences* 4:177–192.

Smith, D. E. 2004. Choosing objectives in over-subscription planning. In *ICAPS 2004*, 393–401.

G-value Plateaus: A Challenge for Planning

J. Benton[†] and **Kartik Talamadupula**[†]

Patrick Eyerich[‡] and **Robert Mattmüller**[‡] and **Subbarao Kambhampati**[†]

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

[‡] Department of Computer Science
University of Freiburg
Freiburg, Germany
{eyerich,mattmuel}@informatik.uni-freiburg.de

Abstract

While the string of successes found in using heuristic, best-first search methods have provided positive reinforcement for continuing work along these lines, fundamental problems arise when handling objectives whose value does not change with search operations. An extreme case of this occurs when handling the objective of generating a temporal plan with short makespan. Typically used heuristic search methods assume strictly positive edge costs for their guarantees on completeness and optimality, while the usual "fattening" and "advance time" steps of heuristic search for temporal planning have the potential of resulting in "g-value plateaus". In this paper we point out some underlying difficulties with using modern heuristic search methods when operating over g-value plateaus and discuss how the presence of these problems contributes to the poor performance of heuristic search planners. To further illustrate this, we show empirical results on recent benchmarks using a planner made with makespan optimization in mind.

Introduction

Search space topology has received significant attention in planning; efforts such as those made by Hoffmann (2005) have, for example, identified the problem of h-value plateaus, regions of the search space where h-values do not change. A related, but less recognized problem within the planning community are g-value plateaus, in which search operations do not increase the g-value over large regions. While h-value plateaus occur because of the imperfect or myopic nature of heuristics, g-value plateaus can trace their roots to a more basic mismatch between the search operations and objectives. Specifically, the children (and descendants) of a search node may not necessarily have a higher g-value than the node. There are many objectives in planning for which standard formulations tend to lead to g-value plateaus. Perhaps the poster child of such objectives is *makespan* minimization using normal state-space (or decision-epoch (Cushing et al. 2007)) search formulations. It is easy enough to see that the operation of adding an action to a plan does not necessarily increase its makespan.[1]

[1] Another example is partial order planning search, where many operations, such as establishment, do not increase cost.

At first glance, g-value plateaus seem to run afoul of the "positive edge cost" assumption that is the textbook sufficiency condition for the completeness/termination of A^* search. Fortunately, while the absence of g-value plateaus is sufficient to guarantee completeness, their presence does not necessarily lead to incompleteness as long as there are no infinitely long bounded-value (zero-cost) paths. This is often of little consolation however as g-value plateaus can, and do, significantly inhibit A^*-search especially when coupled with non-pathological heuristics (i.e., where $h(s) < h^*(s)$). This issue is by no means unique to planning: the work on treewidth computation using best-first search (Dow and Korf 2007) also highlights problems similar to those encountered in makespan minimization (although they do not connect it to the broader g-value plateau problem).

In this paper, we aim to call the community's attention to the problem of g-value plateaus and the need for techniques to handle them. In order to do this effectively, we need to answer two natural questions: (i) why has the planning literature not paid much attention to the deleterious effects of g-value plateaus? and (ii) just how much of an effect do g-value plateaus have on the efficiency of the underlying search? In the following, we will answer both these questions, specifically in the context of finding plans of low *makespan*. For the first, we shall show how existing systems may overlook the presence of g-value plateaus either (a) because they have usually focused on multi-objective search (e.g., Sapa (Do and Kambhampati 2003)) where the search operations are not mismatched with respect to at least one of the objectives or (b) because search space representations must attribute a portion of the minimum cost through a state to the h-value (as against the g-value) to ensure certain properties hold. Turning to question (ii), we will quantify the effect of g-value plateaus on search both in theoretical and empirical terms. In the case of optimal planning we show that g-value plateaus will necessarily cause expansion of all states within the plateau that lead to optimal solutions. For suboptimal planning, we will state criteria for when g-value plateaus can force state expansions and empirically confirm that g-value plateaus can cause poor performance. We conclude the paper with some suggestions and show how "pseudo-multi-objective" search that looks at "cost" objectives even when the main focus is makespan can improve performance without significantly degrading plan quality.

Case Study: Minimizing Makespan

It can be shown that typical temporal state-space planners working to optimize makespan can suffer from plateaus over the objective function. However, this fact may be overlooked at first glance due to their state space models and how they attribute h- and g-values. We present an evaluation of current approaches that shows how plateaus over the objective function can appear.

Temporal Planning Background

Many existing forward and regression heuristic search temporal planners use a decision epoch time point as their g-value.[2] In usual forward chaining search methods (Bacchus and Ady 2001; Do and Kambhampati 2003; Eyerich, Mattmüller, and Röger 2009), this is the "current time" for actions to begin execution; the regression planner TP4 (Haslum and Geffner 2001), on the other hand, uses a different style of search where the g-value represents the time from the goal to a state's decision epoch. This entails keeping time points of where "currently executing" actions end (progression) or begin (regression). In the following, we call these *action time points* to simplify discussion.

State Evaluation

A usual problem for makespan optimization in state space temporal planning relates to adding long actions that must run in parallel with a series of shorter actions to obtain a high quality solution. Often in these cases, many permutations of the shorter actions can be considered. That is, there is long action giving a constant minimum bound on the makespan for a state, a value different than the current decision epoch time point. In the usual A^* evaluation function of $f(s) = g(s) + h(s)$ for a state s, this constant bound can be on either $g(s)$ or $h(s)$. In terms of the discussed planning methods, we call the decision epoch $g_t(s)$ (with a corresponding heuristic $h_t(s)$) and distinguish it from the known minimum makespan, which we call $g_m(s)$. More formally, given a state s and a maximum distance action time point from $g_t(s)$, $t_{max}(s)$ we define $g_m(s) = \max(g_t(s), t_{max}(s))$ (see Figure 1). Note that we cannot (and should not) easily use this view of state evaluation in practice, as it can produce complications in representing the state space (for instance, $h_m(s) = 0$ is not enough for a goal test). Ultimately, since $h_t(s)$ defines a minimum value that can be found with constant computational time, we can perform this transfer of values from h to g. For this reason the transfer serves to simplify discussion and analysis to considering only g-value plateaus whenever such minimum bounds exist in the calculation of h.

Plateaus on g

In temporal planning problem benchmarks, adjacent search states with equal makespan (as defined by g_m) can occur in abundance since it is often possible to execute extraneous actions in parallel with useful ones without changing the final plan makespan, leading to g-value plateaus. Informally, g-value plateaus are a tree structure within the search space where the evaluation values of states do not change from ancestor to descendant. They are inherently *local* in nature. We discuss plateaus on g in both the optimal and suboptimal contexts but first give them a more formal definition.

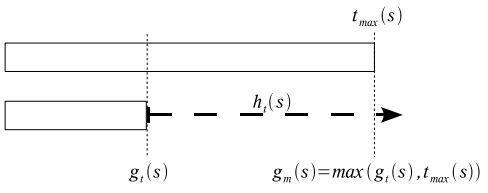


Figure 1: A state s in a (progression) temporal heuristic search planner.

Definition 1. *A g-value plateau with respect to state s is the partial subtree $T[s]$ where s is the root and $T[s]$ includes all immediate descendants s' where $g(s) = g(s')$ and all states in the g-value plateau $T[s']$.*

In the worst case, when all states in the g-value plateau $T[s]$ have a constant heuristic value, upon expanding a state s, A^* will eventually expand all states $s' \in T[s]$. This implies that better heuristics or tie-breaking rules may significantly improve matters. It turns out that this is not always enough. In optimal planning, multiple paths to equally valued states can cause expansion of large portions of the g-value plateau.

Theorem 1. *Given a g-value plateau $T[s]$ (of a state s) that does not contain a goal state and an admissible but non-pathological heuristic function h, upon expanding s during the search, A^* will eventually expand all state $s' \in T'[s]$ before reaching a goal state, where $T'[s] \subseteq T[s] \setminus \{s\}$ is the set of states that can be expanded to an optimal path.*

Proof. Let g^* be the optimal solution value. For any state $s' \in T'[s]$: $h^*(s) = h^*(s')$, since $g(s) + h^*(s) = g(s') + h^*(s') = g^*$ and $g(s) = g(s')$. As h is an admissible and non-pathological heuristic function, $h(s') < h^*(s')$ which implies $f(s') = g(s') + h(s') = g(s) + h(s') < g(s) + h^*(s') = g(s) + h^*(s) = g^*$. Therefore, A* must expand s' before finding an optimal solution. □

A similar problem also occurs in the case of finding suboptimal plans using any heuristic (e.g., inadmissible or weighted heuristics) in the A^* framework. Specifically, we can characterize a portion of the g-value plateau that must be explored.

Proposition 1. *Consider a path $P = (s_0, \ldots, s_g)$ from an initial state s_0 to a solution state s_g found with an A^* search using an (inadmissible) heuristic h. Let s_i be a state in the path such that its g-value plateau, $T[s_i]$, does not contain s_g, s_j $(j > i)$ be the first state s.t. $g(s_j) > g(s_i)$ (and therefore is outside the plateau), and $s_{max} = \arg\max_{s \in (s_j, \ldots, s_g)}\{f(s)\}$. Then, upon expanding*

[2] We are focusing on planners that handle temporally "simple" models, i.e., those that cannot generally handle problems with *required concurrency*.

s_i, before reaching the goal state A^* will have eventually expanded all states s in the plateau that can be reached from s_i through at least one path with all *states s' s.t. $f(s') < f_{max} = f(s_{max})$.*

Proof. As all the states s' along the path from s_i to s satisfy $f(s') < f_{max}$, they will be pulled off the queue before the state s_{max}. □

This proposition implies that there is a portion of the g-value plateau with states s where $h(s) < h_{max} = f_{max} - g(s_i)$ that eventually must be explored. In other words, the heuristic function must return a value greater than the bound h_{max} in order to prune this search space, underscoring that only the h-value can guide search over g-value plateaus.

Empirical Confirmation

We designed experiments to analyze the search spaces of recent temporal planning benchmark problems for g-value plateaus on makespan. To enable this analysis, we designed a new (inadmissible) makespan heuristic (h_m) on top of the context-enhanced additive heuristic used in the Temporal Fast Downward (TFD) planner (Eyerich, Mattmüller, and Röger 2009).

The heuristic uses the same method as TFD for determining sets of actions required to reach the goal. In the makespan heuristic we make use of the causal constraints between actions (assuming TGP semantics (Smith and Weld 1999)) as detected during the extraction of the heuristic plan. These constraints, together with duration constraints between when action begin and end points, are encoded in a Simple Temporal Network (STN). The makespan of the schedule produced by solving the STN is then returned as the heuristic estimate.

Table 1 summarizes the results of the empirical analysis on the 6th International Planning Competition (IPC-2008) domains. Our experiments were run on an Intel Xeon processor running at 2.66 Ghz at a 10 minute time limit running using SuSE Linux. Unlike the original TFD, the planner neither performs an anytime search nor uses "preferred operators". Note that the values r_g and r_f in the table include runs where no solution was found (but a sample of the search space could still be taken).

We define the function $g_c(s)$ as the sum of all action durations chosen at s, $h_c(s)$ as the heuristic on the sums of durations on the "actions-to-go", $h_m(s)$ as our new heuristic, and $g_m(s)$ as described previously (i.e., the makespan of s up to the longest running action). The results show remarkably large portions of the search space with g-value plateaus on makespan in all domains where $f_m = g_m + h_m$ is used. Notice that, except in openstacks-adl and openstacks-strips, f-value plateaus increase significantly with g-value plateaus (an expected result). Contrasting with $f_c = g_c + h_c$, it is apparent that using summed durations improves coverage of problems solved significantly, though the quality of the solutions is reduced. Note that it is possible to get equal g-values on parents and children using g_c in our search with the "advance time" operation, which does not add any actions.

Steps Toward a Solution

By now we have accomplished the main aim of this paper, which is to bring the challenge to the foreground. In this section, we discuss steps towards handling this challenge and share results on one promising idea.

The problem with g-value plateaus is that they can induce search that is worse than standard breadth-first search, a phenomenon that can occur in A^* search with any non-uniform cost values on transitions. g-value plateaus may be seen as a special case of this and offer a step in identifying such "problem" regions of the search space. For instance, one possibility of handling these situations is to find "exit points" to any state beyond the g-value plateau. While it is generally impossible to know f_{max}, we can at least identify g-value plateaus, areas we know have strong dependency on the behavior of the heuristic. In particular, we consider finding a set of exit points that are diverse in their potential reachability across a plateau (c.f., Srivastava et al. (2007)). Of course, such techniques would likely need to be *anytime* in nature for complete and optimal planning.

Another possibility is to try to remove the plateaus by using equivalence classes between states. Such analysis would likely be domain-dependent but if done properly may collapse g-value plateaus. Related to this, in planning for makespan, we can consider using causal analysis to avoid adding extraneous actions. However, this is impractical in general (though approximation methods may be used).

One may consider the idea of adding a small increase in g-values between a parent and child when there would otherwise be zero cost. Indeed, this approach has been applied in the case of planning with action costs (Richter and Westphal 2008; Keyder and Geffner 2008; Benton, Do, and Kambhampati 2009). However, this is unlikely to succeed in general cases. Small increases, given a large enough f_{max}, would exhibit the same behavior. Instead, we study techniques inspired by Sapa and Temporal Fast Downward (TFD) in the temporal planning setting. These planners found success when performing "pseudo-multi-objective" searches.

We developed an approach that applies an additional heuristic cost related both to the makespan (the objective function with g-value plateaus) and the number of search operations left. Specifically, we apply a weighted "cost" heuristic value, $h_c(s)$, which sums the durations of $h_m(s)$ (i.e., the "makespan-to-go") and use the evaluation function $g_m(s) + h_m(s) + w * h_c(s)$. As we will see next, this technique finds some success. The solution differs from those of Sapa and TFD in that we include a state evaluation over makespan while adding a cost evaluation on duration, whereas Sapa uses makespan to calculate its fundamentally cost-based heuristic and TFD only uses the sum of durations as an estimate of makespan. From this perspective the technique is more related to the revised dynamically weighted A^* approach by Thayer and Ruml (2009). While such techniques improve performance, they may not work well in every domain with g-value plateaus, such as problems with known solution depths (for example, best-first search for treewidth (Dow and Korf 2007)).

	$\mathbf{f_c = g_c + h_c}$				$\mathbf{f_m = g_m + h_m}$				$\mathbf{f_{mw} = g_m + h_w}$			
Domain	r_g	r_f	cov	qual	r_g	r_f	cov	qual	r_g	r_f	cov	qual
crewplanning-strips	0.03	0.55	11	6.82	0.98	0.83	4	4.00	0.95	0.09	12	11.99
elevators-numeric	0.06	0.03	4	2.41	0.57	0.27	2	2.00	0.48	0.05	4	3.78
elevators-strips	0.07	0.05	3	1.70	0.53	0.25	3	2.98	0.44	0.04	4	3.92
openstacks-adl	0.15	0.89	30	17.80	1.00	0.88	8	7.58	0.92	0.02	30	28.00
openstacks-strips	0.14	0.88	30	17.12	0.67	0.29	27	20.93	0.71	0.06	30	25.39
parcprinter-strips	0.16	0.08	12	5.68	0.90	0.37	6	5.31	0.76	0.09	6	5.00
pegsol-strips	0.17	0.09	25	18.77	0.85	0.25	22	21.15	0.82	0.08	26	24.04
sokoban-strips	0.28	0.16	11	10.18	0.78	0.32	10	10.00	0.77	0.10	10	9.83
transport-numeric	0.23	0.06	2	1.26	0.74	0.48	3	3.00	0.58	0.09	3	2.78
woodworking-numeric	0.08	0.12	18	14.30	0.72	0.26	18	16.91	0.55	0.04	19	17.64
overall	0.09	0.21	146	96.04	0.84	0.50	103	93.86	0.68	0.07	144	132.37

Table 1: A comparison of the aggregate fraction of zero-cost search operations (r_g), the fraction of f-valued children with equal value to parent (r_f), problems solved (cov) and plan quality according to the IPC-2008 measure (qual, where higher is better compared against the IPC "reference" plans when available) on the IPC-2008 benchmark domains in the temporal satisficing track on $f_c = g_c + h_c$, $f_m = g_m + h_m$ and $f_{mw} = g_m + h_w$.

Empirical Evaluation

In addition to confirming the existence of and problems with g-value plateaus, we ran experiments to test our simple improvement that combines a makespan and weighted cost heuristic using $h_w = h_m(s) + 0.2 * h_c(s)$. With these functions we ran A^* as before but over the evaluation function $f_{mw}(s) = g_m(s) + h_w(s)$.

Our experiments on f_{mw} show that, despite the g-value plateaus that still remain in the search space, performance improves dramatically from f_m (see Table 1). In fact, in most instances, openstacks-adl in particular, the planner appears to be achieving the best of both worlds, with a high number of problems solved with high quality (and our total quality reflects this increase). Also, by adding a weighted h_c, in most cases the search spaces have a significant decrease in f-value plateaus compared with f_c and f_m.

Conclusion

This paper points out the problem of g-value plateaus and shows how they can lead to poor search performance. We focused particularly on planning for quality solutions using makespan as a prime example of where the problem occurs and studied one solution to this problem. Our empirical results show that, when facing g-value plateaus, we can improve performance significantly using a "pseudo-multi-objective" method that includes a heuristic on a related, but different objective. However, this approach is not optimal and we believe that techniques that focus on identifying g-value plateaus may yield more principled and general approaches compatible with both optimal and suboptimal search.

Acknowledgments: We extend our thanks to Tuan Nguyen, William Cushing and Rong Zhou for their helpful discussions. Many thanks also go to Wheeler Ruml, Jordan Thayer, Sofia Lemons and the anonymous reviewers for their excellent comments. This research is supported in part by ONR grants N00014-09-1-0017, N00014-07-1-1049, the NSF grant IIS-0905672, by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS) and by the EU as part of the Integrated Project CogX (FP7-ICT-2xo15181-CogX).

References

Bacchus, F., and Ady, M. 2001. Planning with resources and concurrency: A forward chaining approach. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, 417–424.

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

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. S. 2007. When is temporal planning really temporal? In *Proceedings of the 20th IJCAI*, 1852–1859.

Do, M., and Kambhampati, S. 2003. Sapa: A scalable multi-objective metric temporal planner. *Journal of Artificial Intelligence Research (JAIR)* 20:155–194.

Dow, P. A., and Korf, R. E. 2007. Best-first search for treewidth. In *Proceedings of the 22nd Conference on Artificial Intelligence*.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the context-enhanced additive heuristic for temporal and numeric planning. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*.

Haslum, P., and Geffner, H. 2001. Heuristic planning with time and resources. In *Proceedings of the 6th European Conference on Planning*.

Hoffmann, J. 2005. Where 'ignoring delete lists' works: Local search topology in planning benchmarks. *JAIR* 24:685–758.

Keyder, E., and Geffner, H. 2008. The FF(h_a) planner for planning with action costs. In *Proceedings of the International Planning Competition (IPC)*.

Richter, S., and Westphal, M. 2008. The LAMA planner. using landmark counting in heuristic search. In *Proceedings of the IPC*.

Smith, D. E., and Weld, D. S. 1999. Temporal planning with mutual exclusion reasoning. In *Proceedings of the 16th IJCAI*, 326–337.

Srivastava, B.; Kambhampati, S.; Nguyen, T.; Do, M.; Gerevini, A.; and Serina, I. 2007. Domain independent approaches for finding diverse plans. In *Proceedings of the 20th IJCAI*.

Thayer, J. T., and Ruml, W. 2009. Using distance estimates in heuristic search. In *Proceedings of the 19th ICAPS*.

High-Quality Policies for the Canadian Traveler's Problem

Patrick Eyerich and **Thomas Keller** and **Malte Helmert**
Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Georges-Köhler-Allee 52
79110 Freiburg, Germany
{eyerich,tkeller,helmert}@informatik.uni-freiburg.de

Abstract

We consider the stochastic variant of the Canadian Traveler's Problem, a path planning problem where adverse weather can cause some roads to be untraversable. The agent does not initially know which roads can be used. However, it knows a probability distribution for the weather, and it can observe the status of roads incident to its location. The objective is to find a policy with low expected travel cost.

We introduce and compare several algorithms for the stochastic CTP. Unlike the optimistic approach most commonly considered in the literature, the new approaches we propose take uncertainty into account explicitly. We show that this property enables them to generate policies of much higher quality than the optimistic one, both theoretically and experimentally.

Introduction

The Canadian Traveler's Problem (CTP) was introduced by Papadimitriou and Yannakakis (1991) as a path planning problem with imperfect information about the roadmap. It has drawn considerable attention from researchers in AI search (e. g., Nikolova and Karger 2008; Bnaya, Felner, and Shimony 2009) and is closely related to navigation tasks in uncertain terrain considered in the robotics literature (e. g., Koenig and Likhachev 2002; Ferguson, Stentz, and Thrun 2004; Likhachev and Stentz 2006). Informally, the task is to travel from the initial location to some goal location on an undirected weighted graph. This is complicated by the fact that certain roads may be covered by snow and hence be impassable, and the traversability of a road can only be observed from the two incident locations. The weather remains static during the agent's traversal of the graph, so once a road has been observed, its status is known with certainty. Hence, the problem is fully deterministic apart from the initial state uncertainty about which roads are usable.

Many variants of the CTP have been suggested. Papadimitriou and Yannakakis (1991) describe an adversarial setting and a stochastic setting. In the adversarial setting, the objective is to find a policy that minimizes the worst-case ratio between the actual travel cost and the optimal travel cost under perfect information. In the stochastic setting, the status of each road is determined by an independent random choice according to a known probability distribution, and the objective is to minimize expected travel cost (with some subtleties discussed in the next section).

This paper deals with the stochastic CTP, which is the most frequently considered version of the problem and has itself spawned further variants. For example, Nikolova and Karger (2008) describe an optimal algorithm for the stochastic CTP on *disjoint-path* graphs and a recent paper by Bnaya, Felner, and Shimony (2009) studies a variation of the CTP where the status of a road may be sensed remotely, at a cost, and the objective is to minimize the sum of travel cost and sensing cost. A very similar problem to the stochastic CTP where blocking probabilities are associated with graph vertices rather than edges is discussed in the robot path planning community (e. g., Ferguson, Stentz, and Thrun 2004; Likhachev and Stentz 2006).

Although many papers discuss the stochastic CTP, we are not aware of any work that makes a significant attempt at reasoning about the uncertainty that is an integral part of the problem except for studies of special-case graphs (Nikolova and Karger 2008) or instances with very low amounts of uncertainty (Ferguson, Stentz, and Thrun 2004; Likhachev and Stentz 2006). The predominant approach for the general stochastic CTP and related problems is the optimistic policy that always follows the shortest path that *might* still be traversable under the agent's current information, no matter how likely it is for this path to be blocked at some point.

Our main contribution is that we show that taking uncertainty into account in policies for the CTP leads to significant improvements over the optimistic policy. On the theoretical side, we show that while the optimistic policy can be arbitrarily worse than the optimal solution, probabilistic policies based on the UCT algorithm (Kocsis and Szepesvári 2006) converge to the global optimum. On the empirical side, we show the advantages of probabilistic approaches over greedy optimism on a range of benchmark instances.

In the following section, we formalize the problem and discuss some basic properties. We then present four algorithms for the CTP, including the common optimistic approach as well as more sophisticated techniques that take uncertainty into account. This is followed by a theoretical comparison of the approaches and an empirical evaluation, after which we conclude.

The Canadian Traveler's Problem

An instance of the CTP is a 6-tuple $\mathcal{I} = \langle V, E, p, c, v_0, v_\star \rangle$, where

- $\langle V, E \rangle$ is a connected undirected graph (*roadmap*) with vertex set V (*locations*) and edge set E (*roads*),
- $p : E \to [0, 1)$ defines the *blocking probabilities* of roads,
- $c : E \to \mathbb{N}_0$ defines the *travel costs* of roads, and
- $v_0, v_\star \in V$ are the *initial* and *goal* locations.

Roads with blocking probability 0 are called *guaranteed*. (We do not allow blocking probabilities of 1 because they cause technical complications in several places, but they can be equivalently modeled by omitting the respective roads.)

A *weather* for a CTP instance with roads E is a subset $W \subseteq E$ representing the roads that are traversable (not blocked by snow) in that weather. Weather W is called *good* if v_0 and $v_\star$ remain connected when only using roads in W. Otherwise, W is called *bad*.

The algorithmic problem considered in this paper is that of computing a good *policy* for a CTP instance. As usual for problems of acting under uncertainty, policies can be represented as mappings from belief states to actions. In our case, a belief state is given by the agent's current location on the roadmap and its knowledge about traversability of roads: at each decision step and for each road e, the agent knows e to be traversable, knows e to be blocked, or does not have any information about e. While the agent interacts with the environment, its knowledge about road traversability grows monotonically because the weather does not change dynamically.

To illustrate the random choices of the environment and decision steps of the agent that define the belief space of the problem, the following description shows how a particular *run* (a single interaction of the agent with the environment) on instance $\mathcal{I}$ under policy π proceeds:

- Initially, the environment randomly chooses a weather W by independently marking each road e as blocked with probability $p(e)$ and as traversable otherwise. The problem instance is revealed to the agent, but the randomly chosen weather is not. The agent is initially located at v_0.
- At every decision step, all weather information for the agent's current location v is revealed, i. e., the agent observes which of the roads incident to v are blocked.
- If the current agent location v is the goal location, the run is finished. Otherwise, the agent moves to a new location according to its policy. It may only move to locations that are connected to v by a road e which is traversable under the weather W. This incurs a cost of $c(e)$.
- The cost of the run, denoted by $\mathit{cost}(\mathcal{I}, W, \pi)$, is the sum over all costs incurred by the agent's movements.

We are interested in policies of *expected low cost*, i. e., policies that tend to incur a low cost on a typical run. It is tempting to define the cost of a policy simply as the expected value for $\mathit{cost}(\mathcal{I}, W, \pi)$, where the expectation is with respect to the random choice of weather (and possibly further randomization performed by the policy). However, observe that in case of bad weather it is not possible to complete a run, which is most naturally modeled as infinite cost for that run. This implies that if there is a nonzero chance of bad weather (which is the case iff there exists no path from v_0 to $v_\star$ consisting only of guaranteed roads) the expected cost of all policies would be infinite under this definition.

Fortunately, this problem is easy to avoid by instead defining the cost of the policy as the expected cost for all runs with *good weather*, replacing the prior probabilities for the weather by the posterior probabilities under the condition that the weather is good. It is not hard to prove that changing the probabilities in this fashion does not affect a rational agent's decisions. (Put shortly, the important argument is that it is *always* rational for the agent to assume that the weather is good, because the cost of a run in bad weather is infinite in any case, regardless of the agent's behavior.)

We thus define the cost of a policy π for instance $\mathcal{I}$ as

$$\mathit{cost}(\mathcal{I}, \pi) = \sum_{W \subseteq E} P(W) \cdot \mathit{cost}(\mathcal{I}, W, \pi), \qquad (1)$$

where $P(W)$ is the conditional probability that weather W is chosen given that some good weather is chosen.

Due to the exponential number of possible weathers, it is usually impractical to compute the cost of a given policy π according to Eq. 1. In our empirical experiments we will estimate $\mathit{cost}(\mathcal{I}, \pi)$ by sampling.

Reasonable Policies and Upper Bound. Finding optimal policies for the CTP is difficult. Papadimitriou and Yannakakis (1991) showed that the problem is contained in PSPACE and #P-hard, and so far, optimal solutions could only be generated for instances of trivial size.

However, it is not difficult to provide *upper bounds* on the optimal cost, and to find policies that meet these upper bounds. Let N be the number of locations of a given instance. We can divide each run into *phases* where a new phase begins whenever the agent visits some previously unvisited location for the first time. With N locations, there can be at most $N - 1$ such phases in a run. *Within a phase* that starts at location v and ends at location v', the agent only traverses roads on the *known subgraph*, i. e., the graph consisting of only those roads the agent knows to be traversable, by the definition of phases. (New information can only be obtained when reaching a previously unvisited location, ending the phase.)

We can then demand that movements within a phase are performed on *shortest paths* of the known subgraph. We call policies that satisfy this requirement *reasonable*. At the start of the n-th phase, n distinct location have been visited, and hence at most n roads can be traversed by a reasonable policy until a new location is reached, ending the phase. We can thus bound the total number of movements in the run by $\sum_{i=1}^{N-1} i = \frac{1}{2}(N-1)N$, so that the cost of a reasonable policy in good weather is bounded by $\frac{1}{2}(N-1)NC$, where C is the maximal cost of all roads.

Policies for the CTP

We describe four policies for the CTP: one that ignores the blocking probabilities in its movement decisions and three that take them into account.

All four policies can be described in terms of greedy choices with respect to a *cost function* C_π for belief states. When queried for the next move in belief state b, policy π computes the costs $C_\pi(b')$ for all *successor* belief states b' of b and returns the movements that lead to a successor minimizing the sum of $C_\pi(b')$ and the travel cost from b to b'. To enforce reasonable policies, we define successors of b as those belief states which can be reached through a shortest path in the known subgraph that either ends at the goal or at a location where the agent obtains new information. Once a policy has committed to a movement sequence, no new cost values are computed until the sequence has been completed.

The last policy we consider, UCT, does not actually involve separate computations of $C_\pi(b')$ for each successor. Instead, it only computes $C_\pi(b)$, i. e., performs a computation for the *current* belief state, which produces cost estimates for all successors as a side effect. We abstract from this detail in the following discussion.

It is desirable for cost functions to accurately reflect the actual expected cost to goal. In particular, a policy based on the *optimal cost function* C^* produces optimal behavior. Therefore, we will theoretically compare policies in terms of how accurately their cost functions approximate C^*.

Optimism

We begin with the simplest approach, the *optimistic policy* (OPT). The optimistic policy is a very common approach to the CTP (e. g., Bnaya, Felner, and Shimony 2009) and to robotic motion planning in uncertain environments, where many papers focus on efficient implementations of the policy (e. g., Stentz 1994; Koenig and Likhachev 2002).

The optimistic policy is based on what is called the *free space assumption* in the robotics literature: as long as it is *possible* that a given road is traversable, we assume that it is traversable. Formally, the optimistic cost function in belief state b, $C_{\text{OPT}}(b)$, is the distance from the agent location to the goal in the *optimistic roadmap* for b, which is the graph that includes all roads that are known to be traversable in b or unknown in b. Finding shortest paths in the optimistic roadmap is a standard shortest path problem without uncertainty, and hence $C_{\text{OPT}}(b)$ can be efficiently computed.

A sophisticated implementation of the optimistic policy might use algorithms like D* Lite (Koenig and Likhachev 2002) to speed up distance computations, exploiting that over the course of a run, an agent solves a sequence of *similar* path planning problems, allowing reuse of information. Since the focus of this work is on the *quality* of the policy, which is not affected by how C_{OPT} is computed, our implementation simply uses Dijkstra's algorithm.

Hindsight Optimization

The optimistic policy is indeed exceedingly optimistic: its cost estimates are based on the minimum cost to goal in the *best possible weather* given the agent's knowledge. An alternative approach that is less optimistic but still allows us to reduce cost estimation to (a series of) shortest path computations in regular graphs is *hindsight optimization* (HOP).

At each belief state, the hindsight optimization policy performs a sequence of iterations called *rollouts*. The number of rollouts N is a parameter of the algorithm: more rollouts require more time, but tend to produce more stable cost estimates. In each rollout, we first randomly generate a weather according to the blocking probabilities of the CTP instance that is consistent with the agent's knowledge in the given belief state b. In other words, we randomly determine the status of unknown roads using the correct probabilities. If the resulting weather W is bad, the rollout counts as failed. Otherwise, the rollout counts as successful and we compute the distance from the agent's location to the goal in the subgraph of the roadmap that is traversable in W. The hindsight optimization cost estimate $C^N_{\text{HOP}}(b)$ for N rollouts is the average of the computed distances over all successful rollouts.

An alternative and fairly descriptive name for hindsight optimization is *averaging over clairvoyance* (Russell and Norvig 1995). For each weather we consider, we assume that the agent is "clairvoyant", i. e., knows ahead of time which roads are traversable and hence follows the shortest goal path. Since we do not know the actual weather, we *average* over several weathers through stochastic sampling.

Hindsight optimization has recently attracted considerable interest in the stochastic planning community (e. g., Yoon et al. 2008), where it has served as the basis of some highly efficient planning systems. It has also been successfully used for dealing with hidden information in card games, including the one-player game Klondike Solitaire (Bjarnason, Fern, and Tadepalli 2009) and the two-party games bridge (Ginsberg 1999) and Skat (Buro et al. 2009).

Despite these successes, the approach has well-known theoretical weaknesses: it often converges to a suboptimal policy as the number of rollouts approaches infinity. Frank and Basin (2001) give an example of this for the game of bridge, and Russell and Norvig (1995) describe a very simple MDP where HOP fails. In the next section, we give an example of the suboptimality of the HOP policy for the CTP.

Optimistic Rollout

The assumption of clairvoyance is the Achilles heel of the hindsight optimization approach. Our next algorithm, *optimistic rollout* (ORO), addresses this issue by modifying how each rollout is performed. The optimistic rollout policy computes its cost function C^N_{ORO} in the same way as hindsight optimization, by performing a sequence of N rollouts and averaging over cost estimates for successful rollouts.

The difference between the two algorithms is in how the cost estimates of a rollout are computed: in a successful rollout with weather W, rather than using the clairvoyant goal distance, ORO *simulates the optimistic policy* on W and uses the cost of the resulting run as the rollout cost. Hence, in each rollout the agent follows a shortest path in the optimistic graph until it reaches the goal or a road which is blocked in W. In the latter case, it recomputes the optimistic distances based on the new information and follows a new path, iterating in this fashion until it reaches the goal. The total distance traveled then serves as the rollout cost.

Clearly, optimistic rollout is only one representative of a family of *policy rollout* algorithms, as *any* policy could be used in place of the optimistic policy OPT. We choose OPT because it offers a good trade-off between speed and quality.

UCT

The final approach we consider is the *UCT* algorithm (Kocsis and Szepesvári 2006). UCT is a state-of-the-art algorithm for many problems of acting under uncertainty, including playing Klondike solitaire (Bjarnason, Fern, and Tadepalli 2009), which like the CTP is a single-agent problem where the only source of uncertainty is incomplete information about the probabilistically selected initial state.

Similar to the previous algorithms, UCT performs N rollouts, where N is a parameter. As in the ORO algorithm, each UCT rollout computes an actual run from the agent location to the goal for the given weather, without using information that is hidden to the agent, and uses the average cost of successful rollouts as the overall cost estimate $C^N_{\text{UCT}}(b)$. The difference between UCT and ORO is in how the agent's movements during each rollout are determined. While each rollout is independent in ORO, this is not the case in UCT.

Throughout the following description, let b be the belief state on which the UCT policy is queried. A *belief sequence* $\sigma = \langle b, b_1, \ldots, b_i \rangle$ is a sequence of belief states that describes a possible partial rollout starting from b. We define

- $R^k(\sigma)$: the *number of rollouts* among the first k rollouts for belief state b that start with sequence σ, and
- $C^k(\sigma)$: the *average travel cost* to complete these $R^k(\sigma)$ rollouts from σ, i. e., the average cost that is incurred on these rollouts from the end of σ to the goal.

Each UCT rollout starts from belief sequence $\langle b \rangle$ and iteratively adds successor belief states until the goal is reached. Let ρ be an unfinished belief sequence for the $(k+1)$-th rollout which ends in belief state b_i. We must describe how UCT picks the next belief state among the successors $b'_1, \ldots, b'_m$ of b_i. Let ρ_i be the sequence $\langle \rho; b'_i \rangle$, i. e., ρ extended with b'_i. UCT favors successors that led to *low cost* in previous rollouts (where $C^k(\rho_i)$ is low) and have been *rarely tried* in previous rollouts (where $R^k(\rho_i)$ is low). To balance these criteria, which is the classical trade-off between exploitation and exploration, it picks a candidate ρ_i maximizing the *UCT formula* $B\sqrt{\frac{\log R^k(\rho)}{R^k(\rho_i)}} - \mathit{cost}(\rho, \rho_i) - C^k(\rho_i)$, where $\mathit{cost}(\rho, \rho_i)$ is the travel cost from ρ to ρ_i and $B > 0$ is a *bias* parameter of which more will be said shortly. If $R^k(\rho_i) = 0$, the value of the formula is considered to be ∞, so that the first m rollouts starting with ρ visit each successor once. The UCT formula is designed to select each successor arbitrarily often given sufficiently many visits of ρ, yet successors that have been unpromising in the past are chosen increasingly more rarely over time.

Blind vs. Optimistic UCT. Note that the UCT algorithm as described so far does not take into account any problem-specific information that would bias the rollouts towards the goal. We call the resulting policy *blind UCT* (UCTB). Our experimental results will show that UCTB does not perform very well on the CTP; it would require a prohibitively large number of rollouts to converge to a good policy. However, it is possible to slightly modify the basic UCT algorithm to provide it with some guidance towards the goal. Specifically, we implemented the following two modifications that result in the *optimistic UCT policy* (UCTO):

- When extending a partial rollout ρ which has several unvisited successors, break ties in favor of successors with low C_{OPT} value.
- When evaluating the UCT formula, define $R^k(\sigma)$ and $C^k(\sigma)$ as if there had been M additional rollouts for each successor ρ_i, each with cost $C_{\text{OPT}}(b'_i)$, where M is another algorithm parameter.

These modifications guide early rollouts towards promising parts of the belief space while not affecting the behavior in the limit. Similar extensions to UCT have shown great success in the game of Go (Gelly and Silver 2007).

In our experiments, we used a value of $M = 20$, which was determined empirically. We obtained comparable results for other values in the range 5–80, but significantly worse performance for $M = 0$ or $M = 1$.

Bias Parameter. To complete our discussion of UCT, we describe how we choose the bias parameter B which balances exploration and exploitation. The analysis in the UCT convergence proof by Kocsis and Szepesvári (2006) suggests that B should be chosen in such a way that it grows linearly with the optimal cost $C^*(b)$. As the optimal cost is of course unavailable, we estimate it for the $(k+1)$-th rollout by the average cost of the previous k rollouts. (This is undefined for $k = 0$, but B does not affect the choices of the first rollout anyway.) For the UCTO variant, we additionally divide the bias by 10 to further encourage exploitation.

Theoretical Evaluation

We have introduced four different policies for the CTP. (We treat UCTB and UCTO as a single policy in this section, as all results apply equally to both). What are their strengths and weaknesses? How accurately do their cost functions approximate the true cost C^*? Here we present some formal answers to these questions. For space reasons, we only provide proof sketches. We begin with a basic result:

Theorem 1 *As the number of rollouts N approaches ∞, the HOP, ORO and UCT cost functions converge in probability, and the respective policies converge with probability* 1.

Proof sketch: Individual HOP or ORO rollout costs are independent and identically-distributed bounded random variables, so the strong law of large numbers applies. (Boundedness follows from our discussion of reasonable policies.)

The UCT result is covered by the proof of Theorem 2. ∎

In the rest of this section, we denote the cost functions to which the N-rollout cost functions converge with C^∞_{HOP}, C^∞_{ORO} and C^∞_{UCT} and consider the policies in the limit rather than policies based on a finite number of rollouts. Theorem 1 ensures that these notions are well-defined.

To motivate the ideas underlying our main result, the example instance in Fig. 1 illustrates the different pitfalls that OPT, HOP and ORO fall prey to. We assume that ϵ is very small and limit attention to runs where all roads with blocking probability ϵ are traversable and the road with blocking probability $1 - \epsilon$ is blocked.

The optimistic policy is led astray by the cheap but very unlikely path that reaches $v_\star$ via v_6. It would follow the path v_0–v_5–v_6–v_5–$v_\star$, for a total cost of 170.

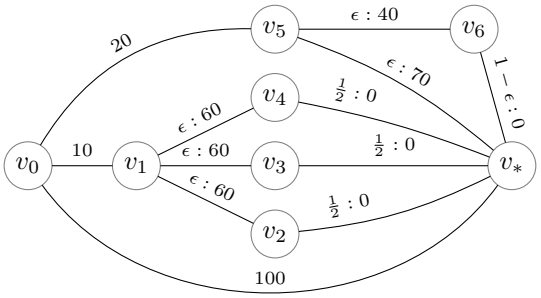


Figure 1: Example with pitfalls for OPT, HOP and ORO. Edge labels $p : w$ denote blocking probability p (omitted for guaranteed roads, i. e., when $p = 0$) and travel cost w.

Hindsight optimization chooses wrongly because there is a high probability of a cheap goal path via v_1 and any of the locations v_2/v_3/v_4, but it is not clear *which* of these three locations to enter. It would assign a cost of 100 to the v_0–$v_\star$ choice, a cost close to 90 to the v_0–v_5 choice (due to path v_0–v_5–$v_\star$) and a cost close to 75 ($= 10 + (\frac{7}{8} \cdot 60 + \frac{1}{8} \cdot 100)$) to the v_0–v_1 choice, hence moving to v_1 first. At v_1 it would realize the suboptimality of its choice and ultimately reach the goal via path v_0–v_1–v_0–v_5–$v_\star$ at cost 110.

Optimistic rollout is fooled by the fact that OPT acts suboptimally in v_5, giving rise to an exaggerated cost estimate for v_5. It would follow the path v_0–$v_\star$ at cost 100.

Finally, UCT converges to the optimal policy, following the path v_0–v_5–$v_\star$ at cost 90. This is a consequence of our main result, which we now present.

Theorem 2 *For all CTP instances $\mathcal{I}$ and belief states b:*

$$C_{OPT}(b) \leq C^{\infty}_{HOP}(b) \leq C^{\infty}_{UCT}(b) = C^*(b) \leq C^{\infty}_{ORO}(b),$$

where UCT *refers to both policy variants. Moreover, there are instances where all inequalities are strict and the ratio between any two different cost functions is arbitrarily large.*

Proof sketch: For UCTB, convergence to the optimal cost function (and hence also to the optimal policy) follows from a slight generalization of Theorem 6 of Kocsis and Szepesvári (2006). The modifications to UCTB that give rise to UCTO do not affect behavior in the limit.

$C^*(b) \leq C^{\infty}_{\text{ORO}}(b)$ holds because each ORO rollout corresponds to an *actual* run of the CTP instance, which cannot have a lower expected cost than the optimal cost C^*.

To prove $C_{\text{OPT}}(b) \leq C^{\infty}_{\text{HOP}}(b) \leq C^*(b)$, let $\mathcal{I}$ be the given instance with road set R and let Π be the set of all policies for $\mathcal{I}$. We can show that for the initial belief state b_0:

$$C_{\text{OPT}}(b_0) = \min_{\pi \in \Pi} \min_{W \subseteq R} \mathit{cost}(\mathcal{I}, W, \pi)$$

$$C^{\infty}_{\text{HOP}}(b_0) = E[\min_{\pi \in \Pi} \mathit{cost}(\mathcal{I}, W, \pi)]$$

$$C^*(b_0) = \min_{\pi \in \Pi} E[\mathit{cost}(\mathcal{I}, W, \pi)]$$

where expected values are w.r.t. the random choice of (good) weather W. The result for b_0 follows from this by simple arithmetic and readily generalizes to all belief states.

To show arbitrary separation between C_{OPT}, C^{∞}_{HOP}, C^* and C^{∞}_{ORO}, we use augmented versions of the "pitfalls" for the respective algorithms exemplified in Fig. 1. ∎

	OPT	HOP	ORO	UCTB	UCTO
20-1	205.9 ± 7	171.6 ± 6	176.3 ± 6	210.7 ± 7	$\mathbf{169.0} \pm 6$
20-2	187.0 ± 5	155.8 ± 3	150.3 ± 3	176.4 ± 4	$\mathbf{148.9} \pm 3$
20-3	139.5 ± 6	138.7 ± 6	134.2 ± 6	150.7 ± 7	$\mathbf{132.5} \pm 6$
20-4	266.2 ± 9	286.8 ± 8	264.2 ± 7	264.8 ± 9	$\mathbf{235.2} \pm 7$
20-5	163.1 ± 7	113.3 ± 6	113.0 ± 6	123.2 ± 7	$\mathbf{111.3} \pm 6$
20-6	180.2 ± 6	142.0 ± 4	134.4 ± 4	165.4 ± 6	$\mathbf{133.1} \pm 3$
20-7	172.2 ± 5	150.2 ± 4	168.8 ± 4	191.6 ± 7	$\mathbf{148.2} \pm 4$
20-8	150.1 ± 6	$\mathbf{133.6} \pm 5$	137.7 ± 5	160.1 ± 7	134.5 ± 5
20-9	222.0 ± 5	177.1 ± 4	176.4 ± 4	235.2 ± 6	$\mathbf{173.9} \pm 4$
20-10	178.2 ± 6	188.1 ± 6	$\mathbf{166.3} \pm 5$	180.8 ± 7	167.0 ± 5
50-1	255.5 ± 10	250.6 ± 9	214.3 ± 7	229.4 ± 12	$\mathbf{186.1} \pm 7$
50-2	467.1 ± 11	375.4 ± 8	406.1 ± 8	918.0 ± 16	$\mathbf{366.5} \pm 7$
50-3	281.5 ± 9	294.5 ± 7	268.5 ± 7	382.1 ± 15	$\mathbf{255.6} \pm 7$
50-4	289.8 ± 9	263.9 ± 7	241.6 ± 7	296.6 ± 12	$\mathbf{230.5} \pm 7$
50-5	285.5 ± 10	239.5 ± 8	229.5 ± 7	290.8 ± 11	$\mathbf{225.4} \pm 7$
50-6	251.3 ± 10	253.2 ± 9	238.3 ± 9	405.2 ± 21	$\mathbf{236.3} \pm 9$
50-7	242.2 ± 9	221.9 ± 7	209.3 ± 7	250.5 ± 11	$\mathbf{206.3} \pm 7$
50-8	355.1 ± 11	302.2 ± 9	300.4 ± 8	462.6 ± 15	$\mathbf{277.6} \pm 8$
50-9	327.4 ± 13	281.8 ± 11	238.1 ± 9	295.2 ± 18	$\mathbf{222.5} \pm 9$
50-10	281.6 ± 8	271.2 ± 7	249.0 ± 7	390.8 ± 15	$\mathbf{240.8} \pm 6$
∅	245.1 ± 2	220.6 ± 2	210.8 ± 2	289.0 ± 3	$\mathbf{200.0} \pm 2$

Table 1: Average travel costs with 95% confidence intervals for 1000 runs on roadmaps with 20 (top) and 50 (bottom) locations. Best results on each graph in bold.

Experimental Evaluation

To evaluate the algorithms empirically, we performed experiments on Delaunay graphs, following the example of Bnaya, Felner, and Shimony (2009). For each algorithm and benchmark graph, we performed 1000 runs to estimate the true policy cost as defined in Eq. 1 with sufficient accuracy.

Main experiment. In our main experiment, we generated random Delaunay graphs with 20–50 locations. Blocking probabilities were chosen uniformly in the range $[0, 1)$, travel costs uniformly from $\{1, \ldots, 50\}$. Initial and goal locations were chosen to be at "opposite ends" of the graph.

We evaluated all policies on these 20 benchmarks, using 10000 rollouts for the probabilistic algorithms. Table 1 shows the outcome of the experiment. The optimistic UCT policy dominates, always providing the best results except for two cases where the difference between UCTO and the best performance is not statistically significant. In addition to UCTO, the HOP and ORO policies also significantly outperform the optimistic policy, clearly demonstrating the benefit of taking uncertainty into account for the CTP. These overall results nicely complement our theoretical results. We conjecture that for some of the graphs where UCTO significantly outperforms the other policies, it reaches a solution quality that is unobtainable for HOP and ORO in the limit.

On average, the quality improvement of UCTO over OPT is larger than 20%, a huge difference. The blind UCT algorithm does not fare well, converging too slowly – a not unexpected result, as the initial rollouts of UCTB have to reach the goal through random walks. The poor performance of UCTB underlines that these benchmarks are far from trivial.

Rollouts and Scalability. To analyze the speed of convergence and scalability of the probabilistic algorithms, we performed additional experiments on individual benchmarks where we varied the rollout number in the range 10–100000.

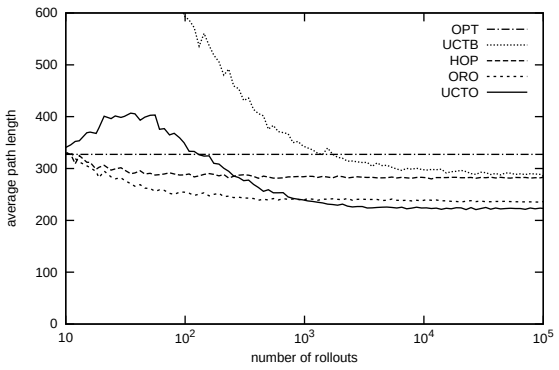


Figure 2: Average travel cost as a function of rollout number for benchmark instance 50-9.

Algorithm	$p = 0.1$	$p = 0.3$	$p = 0.5$	$p = 0.6$
Always	$+30.27\%$	$+32.88\%$	$+39.15\%$	$+30.05\%$
Exp	$+0.43\%$	-0.39%	-6.84%	$+3.30\%$
VOI	$+0.64\%$	$\mathbf{-4.70\%}$	-2.16%	-6.06%
ORO	$\mathbf{-0.12\%}$	-2.12%	-5.76%	-5.11%
UCTO	$+0.13\%$	-2.64%	$\mathbf{-6.95\%}$	$\mathbf{-7.13\%}$

Table 2: Results for CTP with remote sensing (sensing cost 5), reported as average cost differences compared to OPT (called "Never" by Bnaya et al.) for the four different graph classes in the Bnaya et al. benchmark set. Negative numbers indicate improvements. Best performances in bold.

Figure 2 shows the outcome for benchmark graph 50-9. We see that apart from UCTB, the probabilistic algorithms already obtain a better quality than the optimistic policy with only about 100 rollouts, which require very little computation. ORO and HOP begin to level off after about 1000 rollouts, where UCTO still continues to improve.

To provide a reference point for evaluation speed, the UCTO policy with 10000 rollouts requires less than one second per decision on our benchmark instances. We do not have space to report details, but we also performed scaling experiments on benchmarks with up to 500 locations, which show that the advantage of UCTO over the optimistic policy tends to increase on larger instances, while runtime grows slightly faster than linearly in the problem size.

Remote Sensing. In our last experiment, we evaluated the performance of our stochastic algorithms on the benchmark instances of Bnaya et al. These are benchmarks for a different problem, a CTP variant where agents may sense the status of roads from a distance, at a cost of 5. The policies "Always", "Exp", and "VOI" suggested by Bnaya et al. make use of these capabilities. To these policies we compare the solution qualities obtained by our policies when treating the same benchmarks as *regular* CTP instances. Thus, we compare policies that attempt to make use of sensing capabilities intelligently to ones that *never perform remote sensing*. The experimental results (Table 2) show that these never-sensing policies are competitive with the best policies of Bnaya et al.

Conclusion

We investigated the problem of finding high-quality policies for the stochastic version of the Canadian Traveler's problem. In addition to the optimistic policy commonly considered in the CTP literature, we discussed three policies for the CTP which take into account blocking probabilities in their decision-making process.

We studied the convergence properties of these policies and proved a clear ordering between the underlying cost functions. Experimentally, we showed that the new policies, in particular our adaptation of the UCT algorithm, offer significant improvements over the optimistic policy. These improvements are large enough to offer competitive performance to state-of-the art approaches for the CTP with remote sensing even when performing no sensing at all.

In the future, we want to examine if better initialization procedures can further improve the convergence behavior of our UCT-based algorithm. Furthermore, we intend to adapt our algorithms to related problems such as the CTP with remote sensing and to more general problems such as probabilistic planning.

References

Bjarnason, R.; Fern, A.; and Tadepalli, P. 2009. Lower bounding klondike solitaire with Monte-Carlo planning. In *Proc. ICAPS 2009*, 26–33.

Bnaya, Z.; Felner, A.; and Shimony, S. E. 2009. Canadian traveler problem with remote sensing. In *Proc. IJCAI 2009*, 437–442.

Buro, M.; Long, J. R.; Furtak, T.; and Sturtevant, N. 2009. Improving state evaluation, inference, and search in trick-based card games. In *Proc. IJCAI 2009*, 1407–1413.

Ferguson, D.; Stentz, A.; and Thrun, S. 2004. PAO* for planning with hidden state. In *Proc. ICRA 2004*, 2840–2847.

Frank, I., and Basin, D. A. 2001. A theoretical and empirical investigation of search in imperfect information games. *Theoretical Computer Science* 252(1–2):217–256.

Gelly, S., and Silver, D. 2007. Combining online and offline knowledge in UCT. In *Proc. ICML 2007*, 273–280.

Ginsberg, M. L. 1999. GIB: Steps toward an expert-level bridge-playing program. In *Proc. IJCAI 1999*, 584–593.

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In *Proc. ECML 2006*, 282–293.

Koenig, S., and Likhachev, M. 2002. D* Lite. In *Proc. AAAI 2002*, 476–483.

Likhachev, M., and Stentz, A. 2006. PPCP: Efficient probabilistic planning with clear preferences in partially-known environments. In *Proc. AAAI 2006*, 860–867.

Nikolova, E., and Karger, D. R. 2008. Route planning under uncertainty: The Canadian traveller problem. In *Proc. AAAI 2008*, 969–974.

Papadimitriou, C. H., and Yannakakis, M. 1991. Shortest paths without a map. *Theoretical Computer Science* 84(1):127–150.

Russell, S., and Norvig, P. 1995. *Artificial Intelligence — A Modern Approach*. Prentice Hall.

Stentz, A. 1994. Optimal and efficient path planning for partially-known environments. In *Proc. ICRA 1994*, 3310–3317.

Yoon, S.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic planning via determinization in hindsight. In *Proc. AAAI 2008*, 1010–1016.

Creating Dynamic Story Plots with Continual Multiagent Planning

Michael Brenner
Albert-Ludwigs-Universität
Freiburg, Germany
brenner@informatik.uni-freiburg.de

Abstract

An AI system that is to create a story (autonomously or in interaction with human users) requires capabilities from many subfields of AI in order to create characters that themselves appear to act intelligently and believably in a coherent story world. Specifically, the system must be able to reason about the physical actions and verbal interactions of the characters as well as their perceptions of the world. Furthermore it must make the characters act believably–i.e. in a goal-directed yet emotionally plausible fashion. Finally, it must cope with (and embrace!) the dynamics of a multiagent environment where beliefs, sentiments, and goals may change during the course of a story and where plans are thwarted, adapted and dropped all the time. In this paper, we describe a representational and algorithmic framework for modelling such dynamic story worlds, Continual Multiagent Planning. It combines continual planning (i.e. an integrated approach to planning and execution) with a rich description language for modelling epistemic and affective states, desires and intentions, sensing and communication. Analysing story examples generated by our implemented system we show the benefits of such an integrated approach for dynamic plot generation.

Introduction

To tell a story is a challenging task that involves many (if not most) aspects of human intelligence. If the storyteller is an AI system it must effectively simulate a coherent story world and control a number of virtual characters in a manner that seems believable to humans, much as in the Turing Test. Thus, creating believable stories, whether interactively or not, can be considered an "AI-complete" task and requires methods from many subfields of AI, e. g., planning, virtual agents and multiagent systems, reasoning about beliefs and intentions, affective computing, and dialogue systems.

Among these methodologies, planning has probably received the most attention in story generation (Meehan 1977; Lebowitz 1985; Riedl and Young 2004; Si, Marsella, and Riedl 2008). Its relevance results from the structural similarity between plans and plots, both of which describe temporal and causal relations between events. Indeed, temporal-causal coherence, as modeled by the semantics of classical STRIPS-like planning formalisms, can be considered a necessary condition for stories (at least non-postmodern ones).

Yet, Planning research follows a different research agenda than Narrative Intelligence and therefore has developed representations and algorithms that are only of limited use to plot creation. As a result, while plans are used in storytelling systems, many interesting aspects of narrative, e. g., motivation and emotion, must be handled outside the planner. While this is fine in itself, it prevents long-time plotting, usually done by a planner, from taking these aspects into account, e. g. say, planning for motivation changes based on emotional reactions to events. Therefore, in this work, we try to integrate ideas from many of the AI fields mentioned above directly into the planning formalism and algorithm to make it directly suitable for narrative generation.

The paper is structured as follows. We first review relevant work in fields outside narrative research. Then we describe a planning formalism that integrates many of these relevant aspects. We then describe a planning algorithm using this representation, Continual Multiagent Planning, and briefly present our implementation. Analysing a story generated by our program we discuss the benefits of our approach. We conclude with a discussion of its relation to previous work in narrative generation and its possible future uses and extensions.

Related Work I

Our work integrates ideas from several subfields of AI, in particular classical and distributed planning, multiagent systems, knowledge representation and reasoning (mainly about perceptions, beliefs, and emotions) and dialogue systems. Due to space limits, we can only discuss few prototypical inspirations here. At the end of the paper, we will relate our approach to previous work in storytelling research.

Most stories feature several characters and can thus be regarded as *multiagent* environments. To model the beliefs of different agents (and their reasoning about each other) we will integrate multiagent *epistemic* modalities into the planning representation (Fagin et al. 1995). Additionally, similarly to *BDI* models of multiagent cooperation, we will explicitly model the desires and intentions of different agents (Grosz and Kraus 1996). In order to describe how characters gather new information, we will need to model communication and perception as well. Here, we are inspired by approaches to *collaborative dialogue* (Lochbaum 1998) and planning with *sensing* (Petrick and Bacchus 2002).

Algorithmically, our work is based on the intuition that the dynamics of plots are hard to describe naturally with a single plan. Often, plots depend on plans failing or being thwarted, then being dropped or adapted, and finally succeed or fail (where which is which often lies in the eye of the beholder), i.e. as a series of planning and execution cycles performed by the characters. Therefore, what we develop is a *Distributed Continual Planning* (DCP) method (DesJardins et al. 1999; Brenner and Nebel 2009).

Modelling Story Worlds in a Multiagent Planning Formalism

Story worlds usually are multiagent environments. To describe plausible behaviour in such worlds, we need to reason about their dynamics. Thus, we must be able to represent not only the physical actions that agents can perform, but also their perceptual and communicative capabilities as well as their (mutual) beliefs and (joint) goals. To do this in a domain-independent fashion, we use a formal descripition language, the Multiagent Planning Language MAPL (Brenner and Nebel 2009). In this section, we present MAPL informally and discuss the extensions made for this paper.

MAPL is a multiagent variant of PDDL (Planning Domain Definition Language), the de facto standard language for classical planning. Instead of propositions, MAPL uses multi-valued state variables (MVSVs). For example, a state variable *color(ball)* would have exactly one of its possible *domain* values *red, yellow,* or *blue*, as compared to the three semantically unrelated propositions *(color ball red), (color ball yellow), (color ball blue)*. MVSVs have successfully been used in classical planning in recent years,but they also provide distinct benefits when representing story worlds: a) Incomplete knowledge can be expressed by adding an *unknown* value to the domain of a MVSV, b) this idea can be extended to model beliefs and mutual beliefs among characters (Fagin et al. 1995) c) knowledge-seeking actions can be modelled as supplying a yet unknown MVSV value. Thus, sensing and communicative acts are modelled like *wh-questions* (what colour? where? etc.), d) due to the mutual exclusivity of MVSV values, they are well suited for Initial State Revision (Riedl and Young 2006).

In addition to the usual preconditions and effects, MAPL actions have a **controlling agent** who executes the action. MAPL assumes that the controlling agents are fully autonomous when executing actions, i. e. there is no external synchronization or scheduling component. Consequently, an action will only be executed if, in addition to its preconditions being satisfied, the controlling agent *knows* that they hold. Implicitly, all MAPL actions are extended with such **knowledge preconditions**. Similarly, implicit **commitment preconditions** describe, intuitively, that if action a controlled by agent A is included in agent B's plan, this can only be done if A has agreed to perform a.

MAPL models three different ways to affect the beliefs of agents: sensing, copresence, and communication. **Sensor models** describe under which conditions the current value of a MVSV can be perceived. **Copresence models** are multiagent sensor models that induce *mutual belief* about the perceived state variable. Informally, agents are copresent when they are in a common situation where they can not only perceive the same things but also each other. **Communicative acts** in MAPL include declarative statements, questions, requests, and acknowledgments. While declarative statements change the belief state of another agent similarly to sensory actions, the other types of communicative acts affect aspects of the agent that are typically considered static in AI Planning, namely the goals of agents.

MAPL **goals** are first-order formulae, like in PDDL. For storytelling we mostly use them in a specific *conditional* form: By introducting a new MAPL keyword, **"currently"**, we can refer to the current state of the world and the agents' beliefs about it in such a conditional goal formula. MAPL also has **temporary subgoals** (TSGs), which must be satisfied at some point in the plan, but may be violated in the final state. TSGs are represented as explicit symbols in MAPL and thus can be reasoned about by a planner. In particular, they can be active or inactive. This is also true for conditional goals, whose antecedent (condition) may hold in the current state or not. Both kinds of *goal activation* mimic how commitment turns desires into intentions in BDI models of human practical reasoning (Bratman, Israel, and Pollack 1988; Cohen and Levesque 1990). Throughout, the paper we will often refer to activated goals as intentions. **Assertions** are *counterfactual* statements, e. g., "If I knew where to find a sword, I could slay the dragon", that the continual planner may use as temporary subgoals in order to gather missing information necessary to achieving its main goal. Assertions enable the agent to postpone planning for subproblems until it has gained more knowledge, i. e. by partially executing a plan and then switching back to more detailed planning. Thus, assertions encourage proactive goal-driven information gathering (Brenner and Nebel 2009), which for plot generation often seems to be a desirable character trait.

MAPL plans are partially ordered, using different kinds of *causal links*. This is advantageous for plot generation because plans provide explanations for the behaviour of the characters. In contrast to other plan-based approaches we will not use plans directly to represent the whole plot. Since during a continual planning episode usually multiple plans are being generated, executed, and revised, we consider as the plot the execution history of the episode, annotated with relevant (possibly false) beliefs and goals. This **plot graph** comprises a totally ordered "fabula", i. e. the sequence of events that occur. Yet, it also uses explanations from plans and plan monitoring to relate actions to each other by various types of causal links. Such causally annotated histories can be naturally regarded as plots in the sense of E. M. Forster (Forster 1927) and provide ample information for discourse generation, i. e. the presentation of the plot.

Continual Multiagent Planning

How can we generate MAPL plot graphs? The method presented in this section, Continual Multiagent Planning (CMP), is a distributed algorithm, i. e. it describes planning *by* multiple agents, who all have different motivations and beliefs about the world. Being fully distributed, it can be ap-

plied to large interactive environments, e. g., multiplayer online role-playing games. However, it also models planning *for* multiple agents, since even the individual agents' plans may involve several agents if that is necessary to achieve her goals. For storytelling, this means that CMP allows for both character-centric and author-centric plot generation.

The CMP algorithm is shown in algorithm 1 (its subprocedures will only be presented informally). CMP extends Continual Collaborative Planning (CCP), an approach developed in the context of situated human-robot interaction (Brenner and Nebel 2009). Like in CCP, CMP agents deliberately switch between planning, (partial) plan execution, monitoring, plan adaptation and communication. However, CMP agents are not required to be benevolent and always willing to adopt any TSGs proposed by other agents – luckily, since this would prevent conflict, intrigue and drama in the generated plots.

Algorithm 1 CMP AGENT(S, G)

$P = \emptyset$
Received no message:
  if S satisfies G **do**
    return "goal reached"
  else
    $P =$ MONITORINGANDREPLANNING(S, G, P)
  if $P = \emptyset$ **then**
    return "cannot achieve goal G"
  else
    $e =$ EXECUTENEXTACTION(S, P)
    $(S, P) =$ STATEESTIMATION(S, e)
Received (tell-val vx) **from** agent a:
  add $v \doteq x$ to S
Received request(sg) **from** agent a:
  if *cooperative(self, a)* $\notin S$ **then**
    send "will not adopt request sg" **to** a
  $P =$ MONITORINGANDREPLANNING$(S, G \cup sg, \emptyset)$
  if $P = \emptyset$ **then**
    send "cannot execute request sg" **to** a
  else
    add sg to G as temporary subgoal
    send "accept request sg" **to** a

When used for centralised planning (i. e. one planner controls all agents) or when no communication is taking place, a CMP agent alternates between (re-)planning and execution. Subprocedure MONITORINGANDREPLANNING first determines whether a new planning phase should be triggered, either because the agent has sensed an external event that has invalidated its previous plan, or because her goals themselves have changed, or because of an *assertion* that was previously used to advance planning despite missing information and whose detailed planning is now triggered because additional knowledge has become available (Brenner and Nebel 2009). If, for any of the above reasons, planning is triggered the agent replans for those parts of its plan that are no longer valid. The details of the actual (re)planning are irrelevant for the purpose of this paper (any state-of-the-art PDDL planner may be adapted for the purpose); it results in an asynchronous MAPL plan that specifies actions for (possibly) several agents and the causal and temporal relation between them necessary for achieving the planning agent's goal. If the plan involves other agents than the planning agent or those characters she can directly control, the new plan must ensure that they are commited to the (sub)goals their actions contribute to. In the original CCP the new plan would have included maximally one *negotiate_plan(a)* action for each agent a appearing in the plan, since all agents were supposed to be cooperative and their exact role in the plan could freely be discussed in the negotation phase. This is different in CMP, where the planning agent must consider different options for making a commit to a subgoal. This can either be done by negotiation as before, if a is known to be cooperative, or by some form of persuasion, i. e. indirect activation of a conditional goal of a. For example, a hunter may present a bear with a honey comb to raise its appetite and make it walk into a trap. Indirect activation may also be recursive, e. g., when a bank robber r threatens a bank clerk c, thereby making *cooperative(c,r)* true and thus make c open for "requests" in the next step.

As soon as a CMP agent has found (or repaired) a valid plan it enters the execution phase (function EXECUTENEXTACTION). First, an action e on the first level of the plan, i. e. one whose preconditions are satisfied in the current state, is chosen non-deterministically. If the action is executable by the CMP agent himself (this includes communicative actions), it is executed. If not, the planning agent tries to determine whether the action was executed by its controlling agent, i. e. it actively observes changes in the environment relevant for its plans. In both cases, the CMP agent will try to update its knowledge about the world state based on the expected effects and the actual perceptions made (function STATEESTIMATION).

Implementation In our view plots do not only consist of plans, but also of their execution, and the resulting re-evaluation of beliefs, goals, and plans by all character agents. Such an approach can best be implemented in a simulation environment. This is most obvious in *interactive* narrative, where some characters are not controlled by the system, but by human users. Yet simulation is also a convenient way to compute the complex results of several characters acting simultaneously in a common environment, observing and influencing each other constantly, even if controlled by a single "author". Therefore we have implemented MAPSIM, a software environment that automatically generates multiagent simulations from MAPL domains. In other words, MAPSIM interprets the MAPL domain both as the planning domain for each CMP character, but also as an executable model of the environment, so that it can determine the results of the execution of the characters' actions.

Note that while for generating the story analysed in the following section, we invoked MAPSIM non-interactively to emphasise the autonomy of the approach, MAPSIM can also be used interactively. Human users may "play" the role of any character and send MAPL commands to the simulation directly.

Figure 1: A story in the *Quests* domain non-interactively created by MAPSIM.

1 This is a story about Smaug, King Arthur and Prince Valiant.

2 King Arthur was in the castle. 3 The treasure was in the cave. 4 King Arthur rode to the cave. 5 King Arthur saw that Smaug was in the cave.

6 King Arthur rode to the castle. 7 King Arthur saw that Prince Valiant was in the castle. 8 'Please bring me the treasure, Prince Valiant,' King Arthur said. 9 'As you wish, King Arthur,' Prince Valiant replied. 10 'Where is the treasure, King Arthur?' Prince Valiant asked. 11 'The treasure is in the cave, Prince Valiant,' King Arthur said. 12 'Thank you,' Prince Valiant said.

13 Prince Valiant rode to the cave. 14 Prince Valiant saw that Smaug was in the cave. 15 Smaug tried to kill Prince Valiant - but failed! 16 Prince Valiant saw that Smaug was not dead. 17 Prince Valiant killed Smaug.

18 Prince Valiant took the treasure. 19 Prince Valiant rode to the castle. 20 Prince Valiant gave King Arthur the treasure. 21 'Thank you for bringing me the treasure, Prince Valiant,' said King Arthur.

22 King Arthur and Prince Valiant lived happily ever after. Smaug did not.

Analysis of a Worked Example

In order to show the benefits of our integrated approach, we will now analyse a story generated by MAPSIM. It is reproduced in figure 1. During the creation of the story a total of 20 different plans were created by the three characters and the simulation itself. On a 1.6 GHz Intel Pentium and 1GB RAM the *total* planning time was less than 500ms.

As input, MAPSIM was given a formal MAPL domain description and individual goals and beliefs for each of the three characters as well as the true initial world state. The resulting plot graph is (for reasons of space) only shown partly in figure 2. It roughly corresponds to lines 9–15 of figure 1 and gives an impression of the MAPL representation of the plot. The first and last line of figure 1 have no direct correspondence in the plot graph, but are directly produced by MAPSIM: In line 1 the characters are introduced, whereas in line 22 MAPSIM reports on which of the agents have achieved their goal and which have not.[1]

Multimodal interaction Note first that characters' behaviour as generated by CMP seamlessly interleaves physical action, sensing, and communication, e. g. in lines 6–8. Due to the explicit representation of epistemic states and information-gathering actions, the characters will plan which gaps in their knowledge they need to fill in order to further detail their plans. This may result in active observation (as in line 5, where Arthur checks whether the cave is empty) or in information-seeking *subdialogues* (as in lines 10–12).

Plan dynamics When Arthur arrives at the cave, he observes that the dragon, Smaug, is there. Arthur knows that he cannot take the treasure while the dragon is present. Thus, CMP detects, in its monitoring phase, that Arthur's plan has become invalid. Arthur generates a new plan, this time a multiagent plan in which Valiant is supposed to help him get the treasure. Switching to the new plan, Arthur leaves the cave and returns to the castle. We claim that it would be quite difficult to describe the plot so far with a single plan, let alone generate it with a single planner run. Continual planning, on the other hand, seems like the natural way to model how a character reacts to the obstactles she encounters.

A form of *proactive* continual planning is exemplified in lines 8-13. Prince Valiant initially does not know the location of the treasure. Thus he could normally not find a plan to get it and therefore would have to decline Arthur's request. However, the planning domain contains a counterfactual *assertion* stating, informally: "If I knew where the treasure was, I could make a plan to bring it somewhere else". Using this assertion, Valiant is able to deliberately *postpone* part of the planning process and first engage in the short subdialogue of lines 10–12 in order to gather the missing information (Brenner and Nebel 2009). The semantics of assertions is such that, when the missing information becomes available, a new planning phase is triggered. It provides Valiant with a more detailed plan – which he executes until he also encounters Smaug and must again extend the plan to include fighting the dragon. In a different setting ("If I knew where the grail was...") satisfying the replanning condition of the assertion, i. e. closing the knowledge gap, may be the more complex part and constitute most of the resulting plot.

Goal dynamics The standard use of continual planning is to adapt plans to changing conditions in the outside world or an agent's belief about it. However, in real life (and thus in stories) *motivations* change, too. CMP agents can adopt temporary subgoals, e. g., when accepting a request by another agent, as in lines 9 and 12 of figure 1. Such changing goals usually lead to more substantial changes in the plot than mere plan adaptation for the same goal. Only after Arthur's request (line 8), Valiant gets involved in the story at all. In particular, this prompts a nice example of *mixed-initiative* behaviour (line 10), where Valiant immediately asks back to get more information necessary to achieve his new goal.

Characterisation by affective goal activation As noted above, changes in an agent's goals may lead to substantial changes in her behaviour. Indeed, it can be argued that a character is better characterised by her motivations than her (fairly volatile) current state of knowledge. However, a complex character has many motivations. Depending on her internal (or some external) context, motivations may be active (i. e. they drive her current behaviour) or inactive. Such context-dependent *goal activation* allows for a more fine-

[1] Obviously the textual output could be vastly improved, e. g., by proper generation of referring expressions. This output was generated using action-specific English templates that the MAPL domain can be annotated with. This way, plot graphs from arbitrary domains can quickly be rendered into a readable natural-language output.

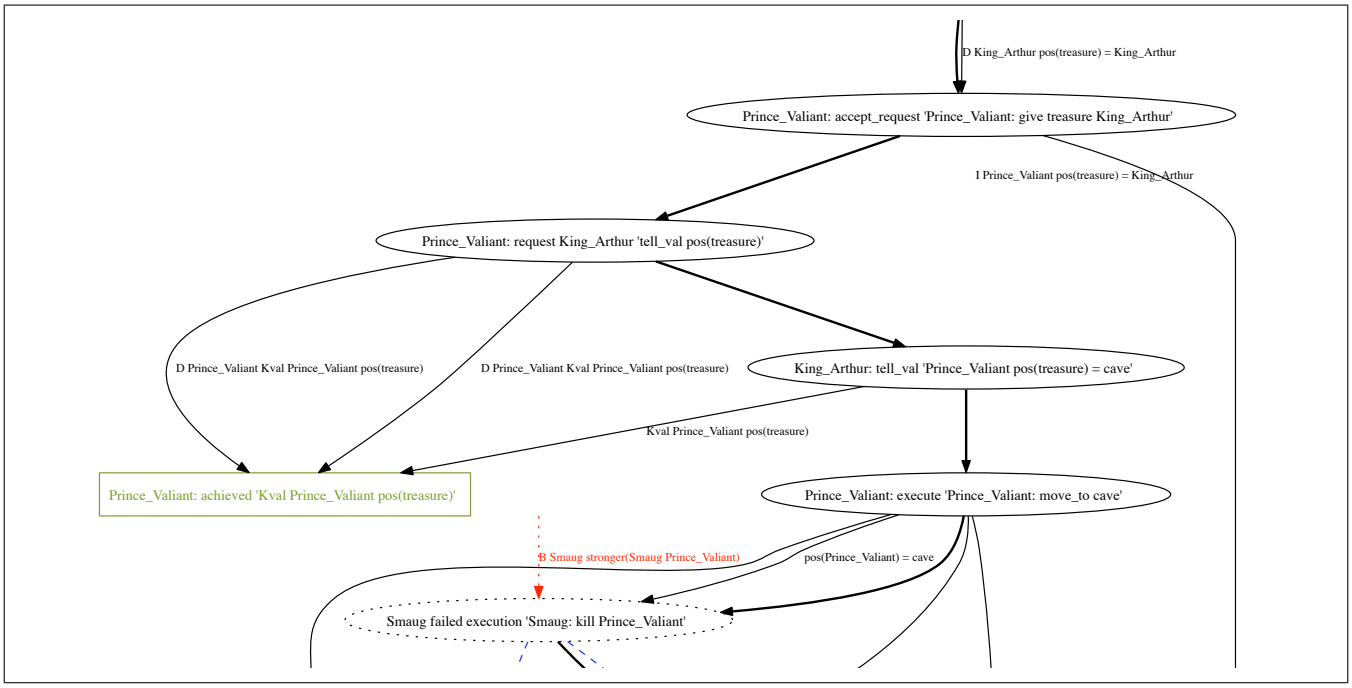


Figure 2: Plot graph (excerpt) for the *Quest* story generated by MAPSIM. Legend: temporal links (bold black), causal links (black), threat prevention links (blue), false belief links (red), TSG achievement (olive box). For clarity, causal links with facts that are true at the beginning have been omitted.

grained characterisation. e. g., in our story, the dragon will only want to kill humans when they have entered its lair.

Using MAPL's *currently* keyword, we can refer to the current state in a goal formula and describe the conditions for goal activation and deactivation.Several such conditional goals can be defined for a character. Together they can define a multi-faceted personality whose concrete intentions may change depending on the current situation, but who will show consistent behaviour in similar situations.

It is important for storytelling that the conditional goals characterising an agent can refer to *emotions* or mental attitudes directed towards other agents and objects, e. g., `angry(a)`, `loves(a,b)`, etc. For example, if the dragon only attacked intruders when angry, but was willing to share the treasure when happy, another story might tell how Prince Valiant charmed the dragon and thus could acquire the treasure without violence. This also opens CMP for use in affective storytelling (Pizzi and Cavazza 2007).

Beliefs, desires, intentions Through MAPL's commitment preconditions CMP enforces characters to commit to a goal/desire before they can actively pursue it, i.e. make it an intention first (Bratman, Israel, and Pollack 1988; Cohen and Levesque 1990). In the multiagent case this means that if character A is not directly controllable by another agent B, B must first somehow persuade A to commit to a new goal before B can assume A's working towards it. In our story, Arthur knows that he cannot "use" Valiant in his plan to get the treasure unless Valiant commits to that goal himself, i. e. makes it an intention of his own. Here, CMP finds a plan for Arthur to achieve this using a simple request (lines 8–9), since in the MAPL description Valiant has been modelled as being cooperative towards Arthur. On the other hand, before CMP could include actions of the dragon into Arthur's plans, it would first have to indirectly activate one of the dragon's own desires.

False beliefs are important for plots, as they result in misunderstandings, in misguided or unsuccessful behaviour. Again, continual planning can be used to reason about the consequences of believing something wrongly. To show this, the example domain is set up such that the "stronger" character always wins a fight. Here, Smaug falsely believes to be stronger than Prince Valiant and attacks him (line 15), which eventually leads to his own death.

Chekhov's gun The plot graph describes which facts and objects are used in the plot. Those facts should be mentioned so that the reader/player can follow the reasoning of the characters. Crucially, the plot graph does not only point to preconditions of actions that characters actually perform, but also to those beliefs never used in an executed plan (because of the plan or the belief becoming obsolete first), but that are necessary to *explain* the changing motivations and plans of the characters.

Related Work II

Having presented our approach to planning for storytelling, we can finally relate it to existing storytelling systems (again, only few representative ones). It should be kept in mind, though, that we do not claim this to be a full sto-

rytelling framework, but only to provide planning representations and algorithms appropriate for being used *inside* such a system. MAPSIM is only a research prototype and a domain-independent testbed to evaluate variants of CMP.

CMP, when used by individual agents in a multiagent simulation like MAPSIM, works as a character-centric approach to storytelling (in contrast to author-centric and story-centric approaches (Mateas and Sengers 1999)) in the tradition of Meehan's Tale-Spin (Meehan 1977). However, since the intentions of several characters are reasoned about explicitly and individually in each plan, it also integrates aspects of author-centric approaches. In this respect, our approach seems closest to Fabulist (Riedl and Young 2004). When CMP is used in the context of a simulation its capability to deliberately devise plans that may fail and to reason about dynamic goals of characters makes it quite suitable to be used for dynamic, interactive storytelling like Emergent Narrative (EM) (Aylett 1999).

Thus, MAPL and CMP integrate features of both author-centric and character-centric approaches. It would be of great interest to evaluate their use in Interactive Storytelling frameworks that strive for a similar integration, e. g., (Si, Marsella, and Riedl 2008; Porteous and Cavazza 2009).

Discussion

The main contribution of this article is an integration of models and methods from a number of AI subfields into a representational (MAPL) and algorithmic (CMP) framework that is well-suited for an "AI-complete" task like storytelling. Providing both a rich representation language for reasoning *about* multi-character environments and a distributed algorithm for planning *by* multiple agents, it combines aspects of both character-centric and author-centric approaches to storytelling. A specific emphasis has been put on enabling the generation of *dynamic* plots, in which beliefs, motivations and character traits may change. We believe that, due to a decade-long focus on planning as an isolated one-shot problem, these dynamics have been insufficiently studied in both planning and storytelling research – despite the fact that plot twists and character development are often said to be what makes a story "interesting".

Interestingness or, more technically, *plot quality* is not an explicit concept anywhere in our approach. Thus it is not surprising that we cannot reliably claim that our approach will generate interesting stories. To this end, we will have to extend the approach by an author or, even better, reader model. In future work, we will investigate how a CMP *author agent* can try to achieve *plot goals* by means of initial state revision (Riedl and Young 2006) and late commitment (Swartjes and Vromen 2007), concepts inspired by game-mastering in pen-and-paper roleplaying games. CMP supports these ideas almost directly, because it can reason about possible MVSV values that have not yet been sensed by any of the characters. In further work, we will consider reader models of plot quality, i. e. subjective, externally defined metrics for plot graphs. Given such a metric, we can iteratively use the author agent to generate series of slightly modified stories, i. e. we can mimic the process of *story revision* in a form of *local search* in the space of plot graphs.

At first glance, creating believable interactions between characters in a story world may seem remote from more "serious" AI topics like robotics. However, this paper was inspired by our work on human-robot collaboration and will in turn most certainly find itself being integrated into robots again now.

Acknowledgments

This research was supported by the EU as part of the Integrated Project CogX (FP7-ICT-2xo15181-CogX).

References

Aylett, R. 1999. Narrative in virtual environments - towards emergent narrative. In *Proc. AAAI Narrative Intelligence Symposium*.

Bratman, M. E.; Israel, D. J.; and Pollack, M. E. 1988. Plans and resource-bounded practical reasoning. *Computational Intelligence* 4:349–355.

Brenner, M., and Nebel, B. 2009. Continual planning and acting in dynamic multiagent environments. *Journal of Autonomous Agents and Multiagent Systems* 19(3).

Cohen, P. R., and Levesque, H. J. 1990. Intention is choice with commitment. *Artificial Intelligence* 42(213–261).

DesJardins, M.; Durfee, E.; C. Ortiz, J.; and Wolverton, M. 1999. A survey of research in distributed, continual planning. *The AI Magazine*.

Fagin, R.; Halpern, J. Y.; Moses, Y.; and Vardi, M. Y. 1995. *Reasoning About Knowledge*. MIT Press.

Forster, E. M. 1927. *Aspects of the Novel*. San Diego: Harcourt.

Grosz, B. J., and Kraus, S. 1996. Collaborative plans for complex group action. *Artificial Intelligence* 86.

Lebowitz, M. 1985. Story-telling as planning and learning. *Poetics* 14:483–502.

Lochbaum, K. E. 1998. A collaborative planning model of intentional structure. *Computational Linguistics*.

Mateas, M., and Sengers, P. 1999. Narrative intelligence. In *Proc. Narrative Intelligence Symposium*.

Meehan, J. R. 1977. Tale-spin, an interactive program that writes stories. In *Proc. IJCAI*.

Petrick, R., and Bacchus, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *Proc. AIPS-02*.

Pizzi, D., and Cavazza, M. 2007. Affective storytelling based on character's feeling. In *Proc. AAAI Symposium on Intelligent Narrative Technologies*.

Porteous, J., and Cavazza, M. 2009. Controlling narrative generation with planning trajectories: the role of constraints. In *Proc. ICIDS*.

Riedl, M. O., and Young, R. M. 2004. An intent-driven planner for multi-agent story generation. In *Proc. AAMAS*.

Riedl, M. O., and Young, R. M. 2006. Story planning as exploratory creativity: Techniques for expanding the narrative search space. *New Generation Computing* 24(3).

Si, M.; Marsella, S. C.; and Riedl, M. O. 2008. Integrating story-centric and character-centric processes for authoring interactive drama. In *Proc. AIIDE*.

Swartjes, I., and Vromen, J. 2007. Emergent story generation: Lessons from improvisational theater. In *Proc. AAAI Fall Symposium on Intelligent Narrative Technologies*.

Coordinated Exploration with Marsupial Teams of Robots using Temporal Symbolic Planning

Kai M. Wurm Christian Dornhege Patrick Eyerich Cyrill Stachniss Bernhard Nebel Wolfram Burgard

***Abstract*— The problem of autonomously exploring an environment with a team of robots received considerable attention in the past. However, there are relatively few approaches to coordinate teams of robots that are able to deploy and retrieve other robots. Efficiently coordinating the exploration with such marsupial robots requires advanced planning mechanisms that are able to consider symbolic deployment and retrieval actions. In this paper, we propose a novel approach for coordinating the exploration with marsupial robot teams. Our method integrates a temporal symbolic planner that explicitly considers deployment and retrieval actions with a traditional utility-based assignment procedure. Our approach has been implemented and evaluated in several simulated environments and with varying team sizes. The results demonstrate that our proposed method is able to coordinate marsupial teams of robots to efficiently explore unknown environments.**

I. INTRODUCTION

The problem of autonomously exploring an environment is one of the fundamental problems for autonomous mobile robots. There are several applications in which robots have been designed to explore their environment such as planetary exploration or in disaster missions. Using a coordinated team of robots instead of a single robot offers advantages such as fault tolerance or performance gains. The problem of multi-robot exploration with homogeneous robots is relatively well understood. Popular approaches to coordinate such teams estimate the cost and the expected information gain of exploring a target location to find optimal assignments of robots to targets [3, 20, 24].

In several exploration scenarios, however, one needs to consider heterogeneous teams of robots with different capabilities. For a task such as the autonomous exploration of lunar craters [1], one can imagine robots that approach the crater and then deploy a specialized robot which descents into the crater. Robots that are able to deploy and retrieve other robots have also been referred to as *marsupial robots* [17]. Such heterogeneous robots typically require to carefully plan deployment and retrieval actions and to properly take into account the different properties of the robots such as their sensor setup, their size and payload, their maximum traveling speed, or the type of terrain they are able to traverse.

This paper addresses the problem of coordinating a team of marsupial robots that explore an unknown environment.

All authors are with the University of Freiburg, Department of Computer Science, D-79110 Freiburg, Germany.

This work has been supported by the German Research Foundation (DFG) under contract number SFB/TR-8 and by the EU as part of the Integrated Project CogX (FP7-ICT-2xo15181-CogX).

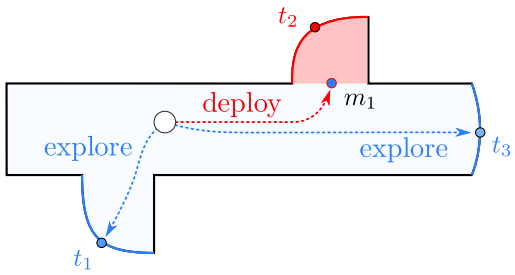


Fig. 1. An exploring robot (white circle) has to choose between three possible actions: explore target t_1, explore target t_3, or deploy a smaller robot at m_1 to let it explore t_2 in the red (dark) area.

From a conceptual point of view, the ability to deploy and retrieve robots by other autonomous robots introduces corresponding symbolic actions which need to be considered in addition to classical exploration actions which typically seek for optimal assignments of robots to target locations (see Fig. 1 for an illustration). Unfortunately, it is not straightforward to map such actions to a cost or utility matrix as they are used by the popular target assignment approaches [3, 16, 19, 24].

The problem of planning and executing actions such as deployment and retrieval in an exploration scenario has previously been approached using manually designed strategies [4, 17, 18]. Such designed hand-crafted strategies, however, are specific to a certain type of environment and it is unclear whether they are able to efficiently coordinate large teams of robots. The contribution of this paper is a novel coordination approach for multi-robot exploration that assigns robots to exploration targets and additionally plans symbolic actions such as deployment and retrieval actions. To achieve this, we integrate a temporal symbolic planner and a traditional path planner for coordinated exploration. In this way, we obtain a more general robot coordination approach that is able to efficiently solve the exploration tasks.

II. RELATED WORK

Several previous approaches consider the task of coordinating the actions of a team of equally equipped (homogeneous) robots exploring an unknown environment. In this setting, the coordination task is often formulated as an assignment problem where the robots are assigned to exploration targets. Different methods have been presented to determine such an assignment. Burgard *et al.* [3] present an iterative assignment method based on the estimated cost of reaching a target and visibility constraints of robots in the

team. Ko *et al.* [16] and Stachniss [19] present approaches that uses the Hungarian method to compute the assignments of frontier cells [23] to robots. Zlot and colleagues [24] propose an architecture in which the exploration is guided by a market economy. They consider sequences of potential target locations for each robot and trade tasks between the robots using single-item first-price sealed-bid auctions. Such auction-based techniques have also been applied by Berhault *et al.* [2] to assign robots to bundles of targets so that synergy effects between targets are taken into account. In a previous work, we present an approach that uses a segmentation of the environment [22]. By assigning robots to unexplored segments instead of frontier targets, a more balanced distribution of the robots over the environment is achieved and the overall exploration time is reduced.

An approach towards cooperation in heterogeneous robot systems is presented by Singh and Fujimura [18]. If a robot is too big to pass through a narrow passage, it informs other robots about this task. Howard *et al.* [14] present an incremental deployment approach that explicitly deals with obstructions, i.e., situations in which the path of one robot is blocked by another. A further heterogeneous system is presented by Grabowski and Navarro-Serment [10]. In this system, however, coordination is performed manually.

Whenever small robots with low traveling speeds or limited power resources are used in a heterogeneous robot team, it is favorable to have larger robots, the *marsupial robots*, transport the smaller ones to avoid a serious penalty in exploration time or power consumption [17]. Denner and Papanikolopoulos present a deployment method for such a marsupial team that explicitly takes power constraints into account [6]. Murphy *et al.* [17] present a physical implementation of a marsupial system and describe heuristics to deploy the micro-rover. Kadioglu and Papanikolopoulos [15] present a further physical implementation. In [4], a team of legged robots is deployed by a carrier robot in a rescue scenario. In all of the previously described exploration systems, deployment and retrieval in marsupial teams is determined by heuristics. In contrast to that, the approach presented in this paper explicitly takes these actions into account when coordinating the exploration.

Domain independent planning is a thoroughly investigated sub-field in artificial intelligence. A classical planning problem consists of a set of state-variables with finite domains, an initial state, a set of actions and a set of goal states. An action is defined by a precondition and its effects, which is a set of variable assignments. A solution for a classical planning problem is then a finite sequence of actions from the initial state to a goal state. There are several efficient planning systems for classical planning problems [12, 13].

When temporal constraints are specified by admitting actions to have variable durations and to be executed concurrently, one refers to that as *temporal planning*. Several efficient approaches for temporal planning have been presented [7, 9]. The predominant approach of solving planning problems is forward search guided by a heuristic using A* or similar algorithms. Most approaches to temporal planning

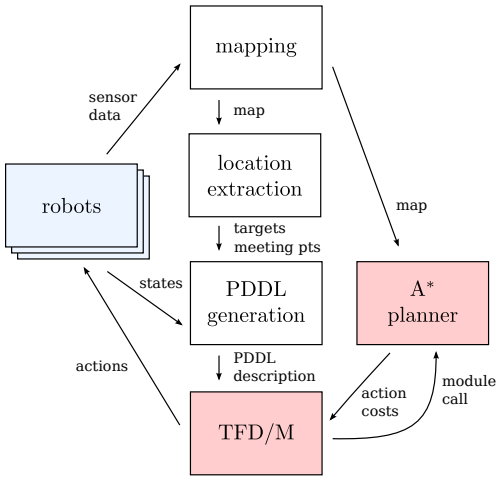


Fig. 2. System overview.

allow the usage of numerical state variables. In contrast to binary and multi-valued state variables, numeric state variables have an infinite continuous value domain. While numeric state variables lead to undecidability even when used in a very limited form [11], they are considered to be of high importance when modeling real world domains.

The work described in this paper builds upon TFD/M [5], a variant of the temporal fast downward planning system [7]. TFD/M is an efficient planning approach that searches directly in the space of time-stamped states. It additionally supports the use of external modules via state variables whose values are calculated by sub-processes during the planning process. By means of sub-processes, we combine temporal planning with path planners traditionally used for multi-robot coordination.

III. COORDINATED EXPLORATION WITH MARSUPIAL TEAMS

Throughout this paper, we assume global and unlimited communication between the robots and employ a centralized approach. Furthermore, all robots are assumed to have known relative positions. To achieve this in our experiments, the robots actually start from the same place in the environment.

A marsupial team consists of two types of robots that explore the environment. We consider n *carrier* robots. Each carrier initially carries m smaller robots called *rovers* which can be deployed and retrieved by the carriers. The key challenge is to generate exploration targets, to plan trajectories for the carriers and rovers, and to schedule deployment and retrieval actions at meeting points in the environment. Especially for efficient retrieval, one needs to consider the time the individual robots need to carry out their actions. This together with the fact that the robots operate in parallel makes our problem a *temporal* planning problem.

A. Overview of the Exploration Framework

The architecture of our exploration system is displayed in Fig. 2. The robots provide the sensor data and states of the platforms (such as their positions, whether they

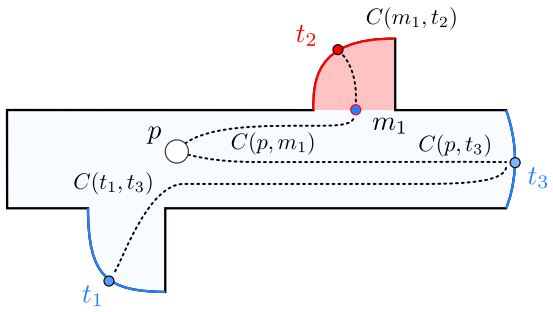


Fig. 3. Example of the costs that have to be considered. Dotted lines illustrate the estimated path costs between the robot pose p and the different target positions t_i, the costs between meeting points m_i and robot pose or targets positions, and costs between target positions. For the sake of better visibility we did not display all costs in this figure.

are docked, etc.) to the centralized coordination system. We use the sensor measurements of the robots to build a grid map distinguishing free, occupied, and unexplored areas. Based on this map, we extract relevant locations in the environment. We then use this information to generate a problem description in the *Planning Domain Definition Language* (PDDL) [8], which serves as input for the temporal planner. In conjunction with a regular path planner such as A*, the temporal planner computes action sequences for the robots that are send to the individual vehicles. The loop depicted in Fig. 2 is constantly executed. Whenever new information about the environment arrives, e.g., new sensor data is perceived or the robots moved, we generate a new plan.

B. Target Locations and Travel Cost

In this work, we model the fact that different robots may have different navigation capabilities and that certain areas of the environment can only be explored by the rovers and others only by carriers. We furthermore assume that the robots are able to determine based on their sensor observations which areas are traversable by which robot, for example based on techniques developed in our previous work [21].

To identify potential target locations, a set of exploration targets T is generated from the partially explored grid map. In addition to this, a set of meeting points M is determined. These meeting points are situated between those parts of the environment that can only be traversed by the rovers and the parts that can only be traversed by the carriers. They are used for deployment and retrieval of the rovers (see Fig. 3 for an illustration).

There are two basic types of actions a carrier can perform: exploring a target or visiting a meeting point to deploy or retrieve a rover (see Fig. 1). While deployment and retrieval are assumed to have constant cost c_{dep}, the cost of traveling between two locations in the environment is defined as the estimated path cost. This cost depends on the path length as well as on the traversability constraints and travel speed of the corresponding robot. Let $type$ be a robot type (here: carrier or rover), x a location in the environment and $t \in \{T \cup M\}$ a target. We define the cost for reaching t as:

$$C_{type}(x,t) = \begin{cases} \text{est. path cost}(x,t)\ , & \text{if robots of type } type \\ & \text{can reach } t \text{ from } x \\ \infty & , \text{ otherwise.} \end{cases} \quad (1)$$

Finally, the exploration task is assumed to be completed as soon as the set of exploration targets T is empty.

C. Formulating the Exploration Problem as a Temporal Planning Problem

A wide range of problem types can be modeled as a general planning problem, ranging from transportation problems and single-player games to combinatorial problems. In recent years, the *Planning Problem Definition Language* (PDDL) [8] has been established as the prevalent planning language. In this paper we use PDDL/M [5], an extension to PDDL allowing for the definition of external modules.

The problem of multi-robot exploration with marsupial robots is a temporal planning problem. The reasons for this are mainly the facts that the actions of the individual robots have an individual duration and that the problem is inherently highly parallel. Especially for the efficient retrieval of rovers, the time the individual robots need to carry out their actions needs to be considered.

To generate a PDDL task description, we need to define (i) the objects involved in the planning process, (ii) the predicates that define the state of the planner, (iii) actions that change the predicates, and (iv) start and goal states.

First, we define what type of objects are involved. In the exploration scenario, possible objects are robots that can be either rovers or carriers and locations that can be meeting points or exploration targets. Fig. 4 (left) illustrates the corresponding PDDL statements. Second, we specify the predicates that define the internal states. The major predicates we use to describe the exploration problem are

`(at ?r - robot ?x - location)`

which describes if the robot r is at position x.

`(on ?e - rover ?c - carrier)`

is used to determine if a rover e is docked to a carrier c. For each target $t \in T$, we also define if it has been explored

`(explored ?t - target)`.

Additionally, we use a numeric fluent

`(num_docked ?c - carrier)`

that contains the number of rovers that are docked to a carrier c.

Third, the actions that change the predicates have to be provided. We need four actions in our setting, namely *dock*, *undock*, *move*, and *explore*. The actions *dock* and *undock* require that the carrier and the rover are at the same meeting point (see `at` predicate). For docking, the number of docked rovers has to be lower than the carrier's capacity and the action changes a rover's state from being at a meeting point to being on a carrier (see `on` predicate).

The other two actions *move* and *explore* model the possible motions of the robots. The *move* action moves a robot to a

meeting point for deployment or retrieval while the *explore* action moves the robot to a target and explores it. For the *move* and *explore* actions, we utilize the module interface [5] of our planning approach to define the duration. Instead of specifying a constant duration or a fixed formula, we call an external module that determines the duration and the actual cost for taking the action. In our setting, the external module is realized by a traditional A* path planner that plans the optimal trajectory of the robot to the given target location based on the current occupancy grid map constructed by the robots so far. Fig. 4 (middle) depicts the PDDL statements that describe the action explore. The term `[pathCost ?r ?s ?g]` represents the call of the external module.

Finally, the initial state of the current planning procedure and the goal state need to be specified. For the situation depicted in Fig. 1 this is exemplified in Fig. 4 (right).

D. The TFD/M planning system

The PDDL description forms the input to the TFD/M planner. Based on this description, the temporal planner computes concurrent action sequences for the robots. TFD/M is a domain-independent progression search planner built on top of the planning system *Fast Downward* [12]. It extends the original system to support durative actions, numeric and object fluents, and external modules.

TFD/M solves a planning problem in three phases: First, the PDDL planning task is translated from its binary encoding into a more concise representation using finite-domain variables. This enables the use of heuristics employing hierarchical dependencies between state variables which leads to an increased search performance. In the second step, efficient internal data structures for the heuristic and the search component are generated. The most important ones are domain transition graphs for each variable that encode how state variables can change their values and the causal graph that represents the hierarchical dependencies between different state variables. Finally, a best-first progression search is performed, guided by a numeric temporal variant of the context-enhanced additive heuristic.

In contrast to many other temporal planning systems, TFD/M does not split the search in an action selection and a scheduling phase but searches directly in the space of time-stamped states. This typically leads to plans of significantly higher quality [7]. Note, however, that due to the inadmissibility of the heuristic evaluation function, the first plan that is generated is not necessarily optimal.

TFD/M does not terminate after a solution was generated, but is implemented as an anytime algorithm. By producing a potentially non-optimal solution quickly, the search space can be pruned to those time-stamped states which can potentially be extended to solutions of a lower overall execution time than the best solution found so far. If all states in the resulting state space are expanded, the produced solution is guaranteed to be optimal.

TFD/M features semantic attachments that are a means of evaluating components of the planning task externally. TFD/M implements this as a module interface for predicates, numerical effects, and durations. In our case, durations of actions are specified as a module call in the planning task description. When expanding actions in the search phase the planner detects these module calls and executes the dynamic library associated with the module call which in turn will retrieve the real costs computed by the A* path planner.

For further details on TFD/M, we refer the reader to our previous work [5, 7].

IV. EXPERIMENTAL EVALUATION

The approach described above has been implemented and evaluated thoroughly using a multi-robot simulation system. The experiments have been designed to show that explicitly planning symbolic action sequences leads to a significantly more efficient coordination approach than using a heuristic extension of previous coordination approaches.

A. Simulation System

To quantitatively evaluate our coordination approach, we developed a simulation system that is able to simulate large teams of marsupial robots. In our current system, we also simulate laser range sensors. Sensor and odometry noise are not considered since we focus on the coordination aspects of the problems. The environment is modeled by a grid map with additional traversability information. The maximum sensor range and traveling speed of carriers and rovers can be specified.

B. Baseline Approach

The baseline approach that we apply to compare our algorithm against is a heuristic extension of a method that assigns robots to target locations based on cost estimates [19]. This approach deploys the rovers heuristically and does not consider them in the target assignment. Thereby the carriers can be assigned to all targets independent of whether they are accessible to them or not. The selection is carried out depending on the estimated costs. If a carrier is assigned to a target that it cannot explore itself it will move to the nearest connecting meeting point and deploy a rover there. This rover will then explore the targets reachable from the meeting point. As soon as it has finished exploring them, it will return to the meeting point. As already mentioned, we assume a limited number of rovers per carrier. If a carrier needs to deploy a rover but has none available, our heuristic requires the carrier to first retrieve a rover.

C. Comparison of Baseline Solution With Our Approach

We evaluated robot teams of varying sizes and different environments have been used in the simulation.

Two of the environments we used to evaluate our approach can be seen in Fig. 5. The office environment resembles a typical office building with two corridors and a number of rooms. Some of the rooms can only be explored by rovers. The maze environment features a central area that can only be explored by rovers but in contrast to the office environment has multiple meeting points that can be used for deployment.

```
(:types
  robot
  carrier rover - robot
  location
  target meeting - location )
(:predicates
  (at ?r - robot ?x - location)
  (on ?e - rover ?c - carrier)
  (explored ?t - target)
  (can_explore ?r - robot ?t - target) )
```

```
(:durative-action explore
  :parameters (?r - robot
     ?s - location ?g - target)
  :duration (= ?duration
     [pathCost ?r ?s ?g])
  :condition (and (at start (at ?r ?s))
     (at start (not (explored ?g)))
     (at start (can_explore ?r ?g)) ... )
  :effect
    (and
      (at start (not (at ?r ?s)))
      (at end (at ?r ?g))
      (at start (explored ?g))
      ... ))
```

```
(:init
  (at robot0 p)
  (on robot1 robot0)
  (can_explore robot0 t1)
  (can_explore robot1 t2)
  (can_explore robot0 t3)
)
(:goal (and
  (explored t1)
  (explored t2)
  (explored t3)
))
```

Fig. 4. Examples for PDDL definitions. Left: definition of the required types and predicates. Middle: definition of the explore action. Right: Example that shows how to specify the current state of the world for the TFD/M planner (see scene shown in Fig. 1).

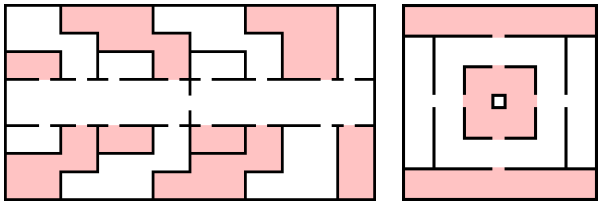


Fig. 5. Our simulated experiments: office (left) and maze (right). White areas can only be traversed by carriers while red (dark) areas can only be explored by rovers.

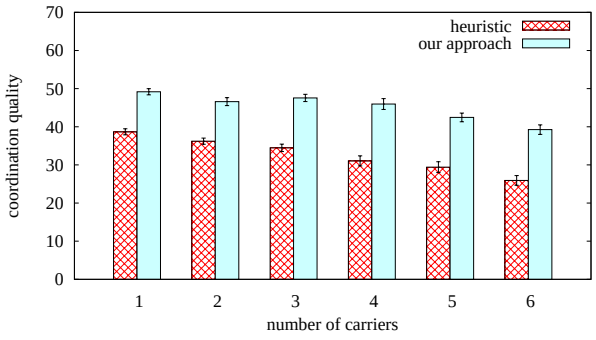


Fig. 7. Coordination quality in the office environment over 30 runs using two rovers per carrier. The higher the value, the better the coordination. The error bars indicate the 95% significance interval. Note that similar results were obtained for the maze environment.

In both environments, we simulated 30 exploration runs using our approach and the baseline method with random initial robot positions. Exploration targets were determined using the frontiers approach and neighboring exploration targets were clustered using visibility constraints similar to the approach proposed by Burgard *at al.* [3].

An overview of the results obtained in these environments is given in Fig. 6. It can be seen that our approach explores the environment significantly faster than the baseline method in all configurations. It can also be seen that using more than three carriers improves the overall exploration time only marginally. The number of rovers for which this effect occurs clearly depends on the structure of the environment and the number of areas that can be explored by rovers only.

As a further benchmark, we computed the coordination quality as defined in [24]:

$$\mathcal{Q} = \frac{1}{A} \sum_{i=1}^{n} d_i, \tag{2}$$

where A is the total area of the environment and d_i denotes the distance traveled by robot i. This measure can intuitively be understood as the area each robot explores per movement.

The results in Fig. 7 show that our approach reaches a significantly higher coordination quality. Especially larger teams of robots are coordinated more efficiently, so that unnecessary movement is avoided. This is especially relevant if the robots have limited power resources so that they can only travel a limited distance until they run out of battery power.

D. Limitations of the Approach

The planning system described in this paper generates sequences of actions for the robots to explore the environment given the current knowledge about the world. While the robots move, their state changes and new information about the environment may be perceived. Therefore, we execute the planning cycle (see Fig. 2) in a continuous loop and use the solution the anytime planner reports. If more than one solution is found we set the timeout to 30 s.

We analyzed our approach with up to 24 robots (6 carriers plus 18 rovers). However, for significantly larger teams, the planning problem becomes large so that the solution reported by the anytime planner after 30 s may be sub-optimal.

V. CONCLUSION

In this paper, we presented a novel approach to coordinate autonomous exploration with marsupial robots. Our approach combines traditional approaches for homogeneous teams that coordinate rovers by solving an assignment problem that maximizes a given evaluation function with a temporal planner that explicitly deals with the deployment and retrieval of small rovers. Our approach has been implemented and thoroughly tested in extensive simulation runs. The experimental results demonstrate that our approach can effectively coordinate large teams of robots and significantly outperforms a handcrafted strategy.

In addition to that, our planning framework adds a substantial degree of flexibility to our system. For example, additional constraints such as power constraints for individual robots can be specified by adding adequate predicates to

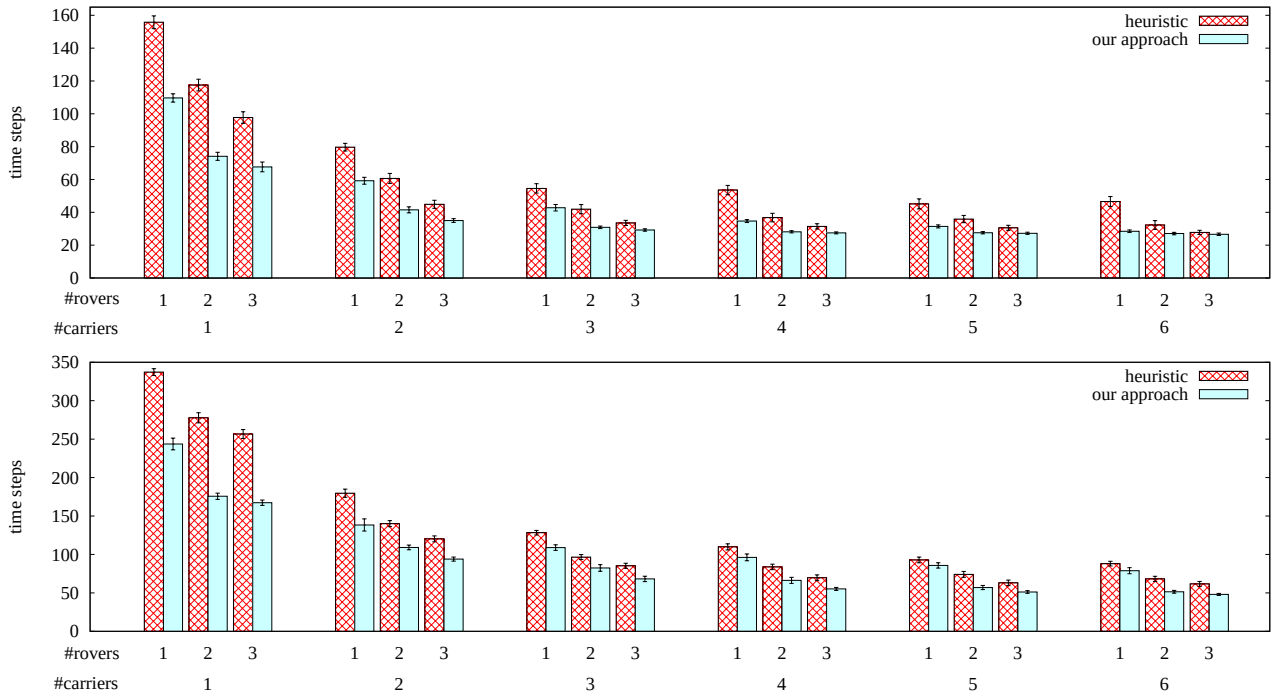


Fig. 6. Exploration time obtained with our approach compared to the heuristic in the maze environment (top) and in the office environment (bottom). The error bars indicate the 95% significance interval.

the problem description. Furthermore, other temporal actions such as recharging batteries or deploying sensor nodes can be integrated in a straightforward way.

REFERENCES

[1] European Space Agency. ESA's lunar robotics challenge website. http://www.esa.int/esaCP/SEMGAASHKHF_index_0.html, 2008.

[2] M. Berhault, H. Huang, P. Keskinocak, S. Koenig, W. Elmaghraby, P. Griffin, and A. Kleywegt. Robot exploration with combinatorial auctions. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 1957–1962, 2003.

[3] W. Burgard, M. Moors, C. Stachniss, and F. Schneider. Coordinated multi-robot exploration. *IEEE Transactions on Robotics*, 21(3):376–378, 2005.

[4] F. Dellaert, T. Balch, M. Kaess, R. Ravichandran, F. Alegre, M. Berhault, R. McGuire, E. Merrill, L. Moshkina, and D. Walker. The Georgia Tech yellow jackets: A marsupial team for urban search and rescue. In *AAAI Mobile Robot Competition Workshop*, 2002.

[5] C. Dornhege, P. Eyerich, T. Keller, S. Trüg, M. Brenner, and B. Nebel. Semantic attachments for domain-independent planning systems. In *Proc. of the Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 114–121, 2009.

[6] A. Drenner and N. Papanikolopoulos. A Framework for Large-Scale Multi-Robot Teams. *Modeling and Control of Complex Systems*, page 297, 2007.

[7] P. Eyerich, R. Mattmüller, and G. Röger. Using the context-enhanced additive heuristic for temporal and numeric planning. In *Proc. of the Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 130–137, 2009.

[8] M. Fox and D. Long. Pddl2.1: an extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)*, 20(1):61–124, 2003.

[9] A. Gerevini, A. Saetti, and I. Serina. An approach to efficient planning with numerical fluents and multi-criteria plan quality. *Artificial Intelligence.*, 172(8-9):899–944, 2008.

[10] R. Grabowski, L.E. Navarro-Serment, C.J.J. Paredis, and P.K. Khosla. Heterogeneous teams of modular robots for mapping and exploration. *Autonomous Robots*, 8(3):293–308, 2000.

[11] M. Helmert. Decidability and undecidability results for planning with numerical state variables. In *Proc. of the Int. Conf. on Artificial Intelligence Planning and Scheduling*, pages 44–53, 2002.

[12] M. Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[13] J. Hoffmann and B. Nebel. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[14] A. Howard, M.J. Matarić, and S. Sukhatme. An incremental deployment algorithm for mobile robot teams. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 2849–2854, 2002.

[15] E. Kadioglu and N. Papanikolopoulos. A method for transporting a team of miniature robots. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 2297–2302, 2003.

[16] J. Ko, B. Stewart, D. Fox, K. Konolige, and B. Limketkai. A practical, decision-theoretic approach to multi-robot mapping and exploration. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 3232–3238, 2003.

[17] R.R. Murphy, M. Ausmus, M. Bugajska, T. Ellis, T. Johnson, N. Kelley, J. Kiefer, and L. Pollock. Marsupial-like mobile robot societies. In *Proceedings of the annual conference on Autonomous Agents*, page 365, 1999.

[18] K. Singh and K. Fujimura. Map making by cooperating mobile robots. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 254–259, 1993.

[19] C. Stachniss. *Robotic Mapping and Exploration*, volume 55 of *STAR Springer tracts in advanced robotics*. Springer, 2009.

[20] C. Stachniss, O. Martinez Mozos, and W. Burgard. Efficient exploration of unknown indoor environments using a team of mobile robots. *Annals of Mathematics and Artificial Intelligence*, 52:205ff, 2009.

[21] K.M. Wurm, R. Kuemmerle, C. Stachniss, and W. Burgard. Improving robot navigation in structured outdoor environments by identifying vegetation from laser data. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

[22] K.M. Wurm, C. Stachniss, and W. Burgard. Coordinated multi-robot exploration using a segmentation of the environment. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2008.

[23] B. Yamauchi. Frontier-based exploration using multiple robots. In *Proc. of the Second International Conference on Autonomous Agents*, pages 47–53, Minneapolis, MN, USA, 1998.

[24] R. Zlot, A.T. Stenz, M.B. Dias, and S. Thayer. Multi-robot exploration controlled by a market economy. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2002.

Partial Weighted MaxSAT for Optimal Planning

Nathan Robinson[†], Charles Gretton[‡], Duc Nghia Pham[†], and Abdul Sattar[†]

[†] ATOMIC Project, Queensland Research Lab, NICTA and
Institute for Integrated and Intelligent Systems, Griffith University, QLD, Australia
[‡] School of Computer Science, University of Birmingham

Abstract. We consider the problem of computing optimal plans for propositional planning problems with action costs. In the spirit of leveraging advances in general-purpose automated reasoning for that setting, we develop an approach that operates by solving a sequence of *partial weighted MaxSAT* problems, each of which corresponds to a step-bounded variant of the problem at hand. Our approach is the first SAT-based system in which a proof of cost-optimality is obtained using a MaxSAT procedure. It is also the first system of this kind to incorporate an admissible planning heuristic. We perform a detailed empirical evaluation of our work using benchmarks from a number of International Planning Competitions.

1 Introduction

Recently there have been significant advances in the direction of optimal planning procedures that operate by making multiple queries to a decision procedure, usually a Boolean SAT procedure. For example, the work of Hoffman *et al.* [1] answers a key challenge from Kautz [2] by demonstrating how existing SAT-based planning techniques can be made effective solution procedures for fixed-horizon planning with metric resource constraints. In the same vein, Russell & Holden [3] and Giunchiglia & Maratea [4] develop optimal SAT-based procedures for *net-benefit* planning in fixed-horizon problems. In that case actions can have costs and goal utilities can be interdependent. Moreover, in the direction of improving the scalability and efficiency of SAT-based approaches in step-optimal (and indeed fixed-horizon) planning, Robinson *et al.* [5] presents an encoding of step-bounded planning problems that shows significant performance gains over previous results. Large performance gains have also been demonstrated where efficient and sophisticated query strategies are employed [6, 7]. Summarising, in the settings of step-optimal and fixed-horizon planning, recent works have demonstrated that SAT-based techniques inspired by systems like BLACKBOX [8] continue to dominate other approaches.

Considering the planning literature more generally, numerous distinct criteria for plan optimality have been proposed. These include: (1) Minimise *makespan* (a.k.a. *step-optimality*); The objective is to find a plan of minimal length. (2) Minimise *plan cost*; Each action has a numeric cost, a plan's cost is the sum of the costs of its constituent actions, and an optimal plan has minimal cost. (3) Maximise *net-benefit*; States (resp. actions) have rewards (resp. costs), and an optimal plan is a sequence of actions executable from the starting state that induces a behaviour of maximal *utility* – These

problems are sometimes called *oversubscribed*, and were recently shown to be equivalent (using a compilation) to the cost-optimising setting [9]. One key observation to be made is that the above optimality criteria are often conflicting. For example, a plan with minimal *makespan* is not guaranteed to be *cost*- or *utility*-optimal. Indeed, in the general case there is no link between the number of plans steps (planning horizon) and plan quality.

Existing SAT-based planning procedures are limited to *makespan*-optimal and *fixed-horizon* settings – i.e., either the objective is to minimise the number of plan-steps, or valid optimal solutions are constrained to be of, or less than, a fixed length. Thus, the use of SAT-based techniques is limited in practice. For example, optimal SAT-based planning procedures were unable to participate effectively at the International Planning Competition (IPC) in 2008 due to the adoption of a single optimisation criteria (cost-optimality). This paper overcomes that restriction, developing COS-P, the fist sound and complete cost-optimal planning procedure based solely on a Boolean SAT(isfiability) procedure. Thus, we open the door to leveraging SAT technology in planning settings with arbitrary optimisation criteria.

The remainder of this paper is organised as follows. We first give an overview of optimal propositional planning with action costs, delete relaxations of that problem, and the partial weighted MaxSAT optimisation problem. We then describe our approach in detail, developing compilations to partial weighted MaxSAT of the fixed-horizon planning problem, and of the fixed horizon problem with a relaxed suffix. Following this we develop our novel MaxSAT solution procedure PWM-RSAT. We then empirically evaluate our approach on planning benchmarks from a number of IPCs. Finally we discuss some related work and propose some interesting directions for future research.

2 Background and Notations

2.1 Propositional planning with action costs

A propositional planning problem with costs is a 5-tuple $\Pi = \langle P, \mathcal{A}, s_0, \mathcal{G}, \mathcal{C} \rangle$. Here, P is a set of propositions that characterise problem states; $\mathcal{A}$ is the set of actions that can induce state transitions; $s_0 \subseteq P$ is the starting state; And $\mathcal{G} \subseteq P$ is the set of propositions that characterise the goal. The function $\mathcal{C} : \mathcal{A} \rightarrow \Re_0^+$ is a bounded cost function that assigns a non-negative cost-value to each action. This value corresponds to the cost of executing the action.

Each action $a \in \mathcal{A}$ is described in terms of its preconditions $pre(a) \subseteq P$, positive effects $eff_\bullet(a) \subseteq P$, and negative effects $eff_\circ(a) \subseteq P$. An action a can be executed at a state $s \subseteq P$ when $pre(a) \subseteq s$. We write $\mathcal{A}(s)$ for the set of actions that can be executed at state s – Formally, $\mathcal{A}(s) \equiv \{a | a \in \mathcal{A}, pre(a) \subseteq s\}$. When $a \in \mathcal{A}(s)$ is executed at s the successive state is $(s \cup eff_\bullet(a)) \backslash eff_\circ(a)$. Actions cannot both add and delete the same proposition – i.e., $eff_\bullet(a) \cap eff_\circ(a) \equiv \emptyset$.[1] A state s is a *goal state* iff $\mathcal{G} \subseteq s$.

Usually any two actions $a_1, a_2 \in \mathcal{A}$ are permitted to be executed instantaneously in parallel at a state provided any serial execution of the actions is valid and achieves an identical outcome. When two actions cannot be executed in parallel we say they

[1] In practice this case is given a special semantics, the details of which shall not be considered further here.

conflict. Supposing non-conflicting actions can be executed instantaneously in parallel, a *plan* π is a discrete sequence of time-indexed sets of non-conflicting actions which, when applied to the start state, lead to a goal state. We say a plan is *serial* (a.k.a. *linear plan*), denoted $\boldsymbol{\pi}$, if each time-indexed set contains one action. Finally, where $\mathcal{A}^i$ is the set of actions at step i of $\pi = [\mathcal{A}^1, \mathcal{A}^2, .., \mathcal{A}^h]$, the cost of π, written $\mathcal{C}(\pi)$, is:

$$\mathcal{C}(\pi) = \sum_{i=1}^{h} \sum_{a \in \mathcal{A}^i} \mathcal{C}(a)$$

A number of different conditions for plan optimality can be defined. In particular, a plan is *parallel step-optimal* if no shorter plan of the same parallel format exists. The definition for *serial step-optimality* is identical, but also respects the condition that a valid plan has only one action executed at each step. A plan π^* is *cost-optimal* if there is no plan π s.t. $\mathcal{C}(\pi) < \mathcal{C}(\pi^*)$. Finally, we draw the reader's attention to the fact that the definition of cost-optimality is not dependent on the plan format.

2.2 The relaxed planning problem

A *delete relaxation* Π^+ of a planning problem Π is an equivalent problem in all respects except the definition of actions. In particular, the set of actions $\mathcal{A}^+$ in Π^+ comprises the elements $a \in \mathcal{A}$ from Π altered so that $\mathit{eff}_\circ(a) \equiv \emptyset$. The relaxed problem has two key properties of interest here. First, the cost of an optimal plan from any reachable state in Π is greater than or equal to the cost of the optimal plan from that state in Π^+. Consequently relaxed planning can yield a useful admissible heuristic in search. For example, a best-first search such as A^* can be heuristically directed towards an optimal solution by using the costs of relaxed plans to arrange the priority queue. Second, although NP-hard to solve optimally in general [10], in practice optimal solutions to the relaxed problem Π^+ are more easily computed than for Π.

2.3 Partial weighted MaxSAT

A Boolean SAT problem is a decision problem, instances of which are typically expressed as a CNF propositional formula. A CNF corresponds to a conjunction over clauses, each of which corresponds to a disjunction over literals. A literal is either a proposition (i.e., Boolean variable symbol) or its negation. Where $\models$ denotes semantic entailment for propositional logic, a solution associated with a formula ϕ is an assignment (a.k.a. valuation) $\mathcal{V}$ of truth values to propositions with the property $\mathcal{V} \models \phi$.

A Boolean MaxSAT problem is an optimisation problem related to SAT. In practice a problem instance is again typically expressed as a CNF, however the objective now is to compute a valuation that maximises the number of satisfied clauses. In detail, writing $\kappa \in \phi$ if κ is a clause in formula ϕ, and taking $\mathcal{V} \models \kappa$ to have numeric value 1 when valid, and 0 otherwise, a solution $\mathcal{V}^*$ to a MaxSAT problem has the property:

$$\mathcal{V}^* = \arg\max_{\mathcal{V}} \sum_{\kappa \in \phi} (\mathcal{V} \models \kappa)$$

A *weighted* MaxSAT problem [11], denoted ψ, is a MaxSAT problem where each clause $\kappa \in \psi$ has a bounded positive numerical weight $\omega(\kappa)$. The optimal solution $\mathcal{V}^*$ to some ψ satisfies the following equation:

$$\mathcal{V}^* = \arg\max_{\mathcal{V}} \sum_{\kappa \in \psi} \omega(\kappa)(\mathcal{V} \models \kappa)$$

Finally, the *partial* weighted MaxSAT problem [12] is a variant of weighted MaxSAT that distinguishes between *hard* and *soft* clauses. Only soft clauses are given a weight.

In these problems a solution is valid iff it satisfies all hard clauses. Therefore we have a notion of satisfiability. In particular, if the *hard* problem fragment of a partial weighted MaxSAT formula is unsatisfiable, then we say the formula is unsatisfiable. The definition of satisfiable follows naturally. An optimal solution to a partial weighted MaxSAT problem is an assignment $\mathcal{V}^*$ that is both valid and satisfies the above equation.

3 COS-P

We now describe COS-P, our planner that operates by iteratively solving variants of n-step-bounded instances of the problem at hand for successively larger n. Solutions to the intermediate step-bounded instances are obtained by compiling them into equivalent partial weighted MaxSAT problems, and then using our own MaxSAT procedure PWM-RSAT to compute their optimal solutions.

COS-P compiles and solves two variants, VARIANT-I and VARIANT-II, of the intermediate instances. Those are characterised in terms of their optimal solutions. Adopting the notation Π_n for the n-step-bounded variant of Π, VARIANT-I admits optimal solutions that correspond to minimal cost plans in the parallel format for Π_n. VARIANT-II admits optimal plans with the following structure. Each has a prefix which corresponds to n *sets* of actions from Π_n.[2] Plans can have an arbitrary length suffix (including length 0) comprised of actions from the delete relaxation Π^+.

Both variants can be categorised as *direct*, *constructive*, and *tightly sound*. They are *direct* because we have a Boolean variable in the MaxSAT problem for every action and state proposition at each plan step. They are *constructive* because any satisfying model and its cost in the MaxSAT instances corresponds to a plan and its cost in the source problem. Critically, our compilations are *tightly sound*, in the sense that every plan with cost c in the source planning problem has a corresponding satisfying model of cost c in the MaxSAT encoding and *vice versa*. This permits two key observations about VARIANT-I and VARIANT-II. First, when both variants yield an optimal solution, and both those solutions have identical cost, then the solution to VARIANT-I is a cost-optimal plan for Π. Second, if Π is soluble, then there exists some n for which the observation of global optimality shall be made by COS-P. Finally, we have that COS-P is a sound and complete optimal planning procedure for propositional problems with action costs.

For the remainder of this section we present the compilation for VARIANT-I and VARIANT-II. In the following section we describe the MaxSAT procedure PWM-RSAT that we developed for use by COS-P.

3.1 VARIANT-I: bounded cost-optimal planning

We now describe a direct compilation of the bounded propositional planning problem with action costs to a partial weighted MaxSAT formula ψ. The source of our compilation is the plangraph. This is an obvious choice because *reachability* and *neededness* analysis performed during construction of the plangraph yields important mutex constraints between action and propositional variables [13]. Such constraints are not deduced independently by modern SAT procedures such as RSAT2.02 [14].

[2] i.e., an n-step plan prefix in the parallel format.

Below, we develop our compilation in terms of a list of 6 Schemata. The first 5 schemata capture the *hard* logical planning constraints, and Schema 6 reflects the action costs. Overall, the schemata we develop below make use of the following propositional variables. For each action occurring at a step $t = 0, .., n-1$ (excluding *noop* actions), we have a variable a^t. We define a fluent to be a state proposition whose truth value can be modified by action executions. For each fluent occurring at step $t = 0, .., n$ we have a variable p^t. Also, we have $make(p) \equiv \{a | a \in \mathcal{A}, p \in \mathit{eff}_\bullet(a)\}$, and $break(p) \equiv \{a | a \in \mathcal{A}, p \in \mathit{eff}_\circ(a)\}$. Below we avoid annotating variables with their time index if it is clear from the context. Lastly, all constraints are hard unless stated otherwise.

1. Goal and start state axioms: We have a unit clause containing p^0 for every $p \in s_0$ and p^n for every $p \in \mathcal{G}$.

2. Precondition and effect axioms: For every action a at each plan step t, we have clauses that require: (i) the action implies its precondition, (ii) the action implies its positive effects, and (iii) the action implies its negative effects:

$$[a^t \rightarrow \textstyle\bigwedge_{p \in \mathit{pre}(a)} p^t] \wedge [a^t \rightarrow \bigwedge_{p \in \mathit{eff}_\bullet(a)} p^{t+1}] \wedge [a^t \rightarrow \bigwedge_{p \in \mathit{eff}_\circ(a)} \neg p^{t+1}]$$

3. Mutex axioms: For every pair of mutex symbols (actions or fluents) p_1 and p_2 at step t, we have a clause: $\neg p_1^t \wedge \neg p_2^t$

4. At least one action axioms: Where $\mathcal{A}^t$ is the set of actions at step t, we have a clause that requires at least one action be executed at step t: $\bigvee_{a^t \in \mathcal{A}^t} a^t$

5. Frame axioms: These constrain how the truth values of fluents change over successive plan steps. For each proposition $p^t, t > 0$ we include the following clauses:

$$[p^t \rightarrow (p^{t-1} \vee \textstyle\bigvee_{a \in make(p)} a^{t-1})] \wedge [\neg p^t \rightarrow (\neg p^{t-1} \vee \bigvee_{a \in break(p)} a^{t-1})]$$

6. Action cost axioms (soft): Finally, we have a set of soft constraints for actions. In particular, for each action variable a^t such that $\mathcal{C}(a) > 0$, we have a unit clause $\kappa_i := \{\neg a^t\}$ with weight $\omega(\kappa_i) = \mathcal{C}(a)$.

3.2 VARIANT-II: n-step with a relaxed suffix

We now describe a direct compilation of the problem Π_n from the previous section, along with the addition of a causal encoding of the delete relaxation, that we make available from step n.[3] From hereon we refer to the latter as the relaxed suffix.

Our encoding of the relaxed suffix is *causal* in the sense developed in [15] for their ground parallel *causal* encoding of propositional planning in SAT. This requires additional variables to those developed for VARIANT-I. In particular, for each fluent p and relaxed action $a \in \mathcal{A}^+$ we have corresponding variables p^+ and a^+. That p_i^+ is true intuitively means: (1) That p_i^n was false (see VARIANT-I), and (2) That $p_i \in \mathcal{G}$, or p_i^+ is the cause of another fluent p_j^+ in a relaxed suffix to the goal. That a^+ is true means that a is executed in the relaxed suffix. We also require a set of causal link variables. These are best introduced in terms of a recursively defined set S^∞ as follows.

$$S^0 \equiv \{\mathcal{K}(p_i, p_j) | a \in \mathcal{A}^+, p_i \in \mathit{pre}(a), p_j \in \mathit{eff}_\bullet(a_i)\}$$
$$S^{i+1} \equiv S^i \cup \{\mathcal{K}(p_j, p_l) | \mathcal{K}(p_j, p_k), \mathcal{K}(p_k, p_l) \in S^i\}$$

For each $\mathcal{K}(p_i, p_j) \in S^\infty$ we have a corresponding variable. Intuitively, if proposition $\mathcal{K}(p_i, p_j)$ is true then p_i is the cause of p_j in the plan suffix.

VARIANT-II includes all schemata from VARIANT-I except the *goal axioms* of Schema 1. We also suppose Schema 6 is now inclusive of a^+ symbols. Additionally we have the following Schemata.

[3] In VARIANT-II goal constraints from Schema 1 are omitted from Π_n.

7. Relaxed goal axioms: For each fluent $p \in \mathcal{G}$ we assert that it is either achieved at the planning horizon n, or using a relaxed action in $\mathcal{A}^+$. This is expressed with a clause:

$$p^n \vee p^+$$

8. Relaxed fluent support axioms: For each fluent p we have a clause:

$$p^+ \rightarrow (\bigvee_{a \in make(p)} a^+)$$

9. Causal link axioms: For all fluents p_i, taking all $a \in make(p_i)$ and $p_j \in \texttt{PRE}(a)$, we have the following clause: $(p_i^+ \wedge a^+) \rightarrow (p_j^n \vee \mathcal{K}(p_j^+, p_i^+))$

This constraint asserts that if action a_1^+ is executed, then its preconditions must be true at horizon n, or be supported by some other action a_2^+ with $p_2 \in \mathit{eff}_\bullet(a_2)$.

10. Causality implies cause and effect axiom: For each causal link variable $\mathcal{K}(p_1^+, p_2^+)$ we have a clause: $\mathcal{K}(p_1^+, p_2^+) \rightarrow (p_1^+ \wedge p_2^+)$

11. Transitive closure and anti-reflexivity axioms: For causal link variable $\mathcal{K}(p^+, p^+)$ we have a unit clause containing that variable negated. For pairs of causal link variables ($\mathcal{K}(p_1^+, p_2^+)$, $\mathcal{K}(p_2^+, p_3^+)$): $(\mathcal{K}(p_1^+, p_2^+) \wedge \mathcal{K}(p_2^+, p_3^+)) \rightarrow \mathcal{K}(p_1^+, p_3^+)$

12. Only necessary relaxed fluent axioms: For each fluent p we have a constraint:

$$\neg p^+ \vee \neg p^n$$

13. Relaxed action cost dominance axioms: Let $\overrightarrow{P}$ be a set of non-mutex fluents at horizon n. Relaxed action a_1^+ is *redundant* in an optimal solution to a VARIANT-II instance, if the fluents in $\overrightarrow{P}$ are true at horizon n and there exists a relaxed action a_2^+ such that: (1) $cost(a_2) \leq cost(a_1)$, (2) $\mathit{pre}(a_2) \backslash \overrightarrow{P} \subseteq \mathit{pre}(a_1) \backslash \overrightarrow{P}$, and (3) $\mathit{eff}_\bullet(a_1) \backslash \overrightarrow{P} \subseteq \mathit{eff}_\bullet(a_2) \backslash \overrightarrow{P}$. For relaxed action a^+ that is *redundant* for $\overrightarrow{P_1}$ and not redundant for any $\overrightarrow{P_2}$, if $|\overrightarrow{P_2}| < |\overrightarrow{P_1}|$ we have a clause:[4] $(\bigwedge_{p \in \overrightarrow{P_1}} p^n) \rightarrow \neg a^+$

The schemata we have given thus far are theoretically sufficient for our purpose. However, in a relaxed suffix most causal links are not relevant to the relaxed cost of reaching the goal from a particular state at horizon n. For example, in a logistics problem, if a truck t at location l_1 needs to be moved directly to location l_2, then the fact that the truck is at any other location should not support it being at l_2 – i.e. $\neg\mathcal{K}(\mathrm{at}(t, l_3), \mathrm{at}(t, l_2)), l_3 \neq l_1$.

The following schemata provide a number of *layers* that actions and fluents in the relaxed suffix can be assigned to. Fluents and actions are forced to occur as early in the set of layers as possible and are only assigned to a layer if all supporting actions and fluents occur at earlier layers. The orderings of fluents in the relaxed layers is used to restrict the truth values of the causal link variables. The admissibility of the heuristic estimate of the relaxed suffix is independent of the number of relaxed layers.

We pick an horizon $k > n$ and generate a copy a^{+l} of each relaxed action a^+ at each layer $l \in \{n, ..., k-1\}$ and a copy p^{+l} of each fluent p^+ at each layer $l \in \{n+1, ..., k\}$. We also have an auxiliary variable $aux(p^{+l})$ for each fluent p^{+l} at each suffix layer $n+1, ..., k$. Intuitively, proposition $aux(p^{+l})$ says that p is false at every layer in the relaxed suffix from n to l.[5]

14. Layered relaxed action axioms: For each layered relaxed action a^{+l} we have a clause: $a^{+l} \rightarrow a^+$

15. Layered relaxed actions only once axioms: For each relaxed action a^+ and pair of layers $l_1, l_2 \in \{n, ..., k-1\}$, where $l_1 \neq l_2$, we have: $\neg a^{+l_1} \vee \neg a^{+l_2}$

[4] In practise we limit $|\overrightarrow{P_1}|$ to 2.

[5] There are no cost constraints associated with the layered copies of relaxed action variables.

16. Layered relaxed action precondition axioms: For each layered relaxed action a^{+l_1} we have a set of clauses: $a^{+l_1} \rightarrow \bigwedge_{p \in \texttt{PRE}(a)} \bigvee_{l_2 \in \{n,\ldots,l_1\}} p^{+l_2}$

17. Layered relaxed action effect axioms: For each layered relaxed action a^{+l_1} and $p \in \texttt{ADD}(a)$ there is a clause: $(a^{+l_1} \wedge p^{+}) \rightarrow \bigvee_{l_2 \in n+1,\ldots,l+1} p^{+l_2}$

18. Layered relaxed action as early as possible axioms: For each layered relaxed action a^{+l_1}, if $l_1 = n$, we have a clause: $a^{+} \rightarrow \bigvee_{p \in \texttt{PRE}(a)} \neg p^{n} \ \vee a^{+n}$

if $l_1 > n$, we add: $a^{+} \rightarrow \bigvee_{l_2 \in n,\ldots,l_1-1} a^{+l_2} \vee \bigvee_{p \in \texttt{PRE}(a)} aux(p^{+l_1}) \vee a^{+l_1}$

19. Auxiliary variable axioms: For each auxiliary variable $aux(p^{+l_1})$ there is a set of clauses: $aux(p^{+l_1}) \longleftrightarrow (p^{n} \wedge \bigwedge_{l_2 \in \{n+1,\ldots,l_1\}} \neg p^{+l_2})$

20. Layered fluent axioms: For each layered fluent p^{+l} we add: $p^{+l} \rightarrow p^{+}$

21. Layered fluent frame axioms: For each layered fluent p^{+l} there is a clause:
$$p^{+l} \rightarrow \bigvee_{a \in make(p)} a^{+l-1}$$

22. Layered fluent as early as possible axioms: For each layered fluent p^{+l_1} there is a set of clauses: $p^{+l_1} \rightarrow \bigwedge_{a \in make(p)} \bigwedge_{l_2 \in n,\ldots,l_1-2} \neg a^{+l_2}$

23. Layered fluent only once axioms: For each fluent p and pair of layers $l_1, l_2 \in \{n+1, ..., k\}$, where $l_1 \neq l_2$, there is a clause: $\neg p^{+l_1} \vee \neg p^{+l_2}$

24. Layered fluents prohibit causal links axioms: For each layered fluent $p_1^{+l_1}$ and fluent p_2 such that $p_1 \neq p_2$ and $\exists \mathcal{K}(p_2^{+}, p_1^{+})$ there is a clause:
$$p_1^{+l_1} \rightarrow (\bigvee_{l_2 \in \{n+1,\ldots,l-1\}} p_2^{+l_2} \vee \neg \mathcal{K}(p_2^{+}, p_1^{+}))$$

4 PWM-RSat

We find that branch-and-bound procedures for *partial weighted MaxSAT* [11, 12] are ineffective at solving our direct encodings of bounded planning problems. Thus, taking the RSAT2.02 codebase as a starting point, we developed PWM-RSAT, a more efficient optimisation procedure for this setting. An outline of the algorithm is given in Algorithm 1. Based on RSAT [16], PWM-RSAT can broadly be described as a backtracking search with Boolean unit propagation. It features common enhancements from state-of-the-art SAT solvers, including conflict driven *clause learning* with *non-chronological* backtracking [17, 18], and *restarts* [19].

Algorithm 1 outlines two variants of PWM-RSAT for solving VARIANT-I and VARIANT-II formulas: lines 5-6 will only be invoked if the input formula is a VARIANT-II encoding. These lines prevent the solver from exploring assignments implying that the same state occurs at more than one planning layer.

Apart from the above difference, the two variants of PWM-RSAT work as follows. At the beginning of the search, the current partial assignment $\mathcal{V}$ of truth values to variables in ψ is set to empty and its associated cost c is set to 0. We use $\hat{c}$ to track the best result found so far for the minimum cost of satisfying ψ^{∞} given ψ^{+}. $\mathcal{V}^{*}$ is the total assignment associated with $\hat{c}$. Initially, $\mathcal{V}^{*}$ is empty and $\hat{c}$ is set to an input non-negative weight bound $\hat{c}^{I}$ (if none is known then $\hat{c} = \hat{c}^{I} := \infty$). Note that the set of *asserting clauses* Γ is initiated to empty as no clauses have been learnt yet.

The solver then repeatedly tries to expand the partial assignment $\mathcal{V}$ until either the optimal solution is found or ψ is proved unsatisfiable (line 4-21). At each iteration, a call to *SatUP*$(\mathcal{V}, \psi, \kappa)$ applies unit propagation to a unit clause $\kappa \in \psi$ and adds new variable assignments to $\mathcal{V}$. If κ is not a unit clause, *SatUP*$(\mathcal{V}, \psi, \kappa)$ returns 1 if κ is

Algorithm 1 Cost-Optimal RSat —- PWM-RSAT

```
1:  Input:
    - A given non-negative weight bound ĉ^I. If none is known: ĉ^I := ∞
    - A CNF formula ψ consists of the hard clause set ψ^∞ and the soft clause set ψ^+
2:  c ← 0; ĉ ← ĉ^I;
3:  V, V* ← []; Γ ← ∅;
4:  while true do
5:      if solving Variant-II && duplicating-layers(V) then
6:          pop elements from V until ¬duplicating-layers(V); continue;
7:      c ← Σ_{κ∈ψ^+} ω(κ)SatUP(V, ψ, κ);
8:      if c ≥ ĉ then
9:          pop elements from V until c < ĉ; continue;
10:     if ∃κ ∈ (ψ^∞ ∧ Γ) s.t. ¬SatUP(V, ψ^∞ ∧ Γ, κ) then
11:         if restart then V ← []; continue;
12:         learn clause with assertion level m; add it to Γ;
13:         pop elements from V until |V| = m;
14:         if V = [] then
15:             if V* ≠ [] then return ⟨V*, ĉ⟩ as the solution;
16:             else return UNSATISFIABLE;
17:     else
18:         if V is total then
19:             V* ← V; ĉ ← c;
20:             pop elements from V until c < ĉ;
21:         add a new variable assignment to V;
```

satisfied by $\mathcal{V}$, and 0 otherwise. The current cost c is also updated (line 7). If $c \geq \hat{c}$, then the solver will perform a backtrack-by-cost to a previous point where $c < \hat{c}$ (line 8-9).

During the search, if the current assignment $\mathcal{V}$ violates any clause in $(\psi^{\infty} \wedge \Gamma)$, then the solver will either (i) restart if required (line 11), or (ii) try to learn the conflict (line 12) and then backtrack (line 13). If the backtracking causes all assignments in $\mathcal{V}$ to be undone, then the solver has successfully proved that either (i) $(\mathcal{V}^*, \hat{c})$ is the optimal solution, or (ii) ψ is unsatisfiable if $\mathcal{V}^*$ remains empty (line 14-16). Otherwise, if $\mathcal{V}$ does not violate any clause in $(\psi^{\infty} \wedge \Gamma)$ (line 17), then the solver will heuristically add a new variable assignment to $\mathcal{V}$ (line 21) and repeat the loop in line 4. Note that if $\mathcal{V}$ is already complete, the better solution is stored in $\mathcal{V}^*$ together will the new lower cost $\hat{c}$ (line 19). The solver also performs a backtrack by cost (line 20) before trying to expand $\mathcal{V}$ in line 21.

5 Experimental Results

We implemented both COS-P and PWM-RSAT in C++. We now discuss our experimental comparison of COS-P with IPC baseline planner BASELINE,[6] and a version of COS-P called H-ORACLE. The latter is given (by an oracle) the shortest horizon that yields a globally optimal plan. Planning benchmarks included in our evaluation include: IPC-6: ELEVATORS, PEG SOITAIRE, and TRANSPORT; IPC-5: STORAGE, and TPP; IPC-3: DEPOTS, DRIVERLOG, FREECELL, ROVERS, SATELLITE, and ZENOTRAVEL; and IPC-1: BLOCKS, GRIPPER, and MICONIC. We also developed our own domain, called FTB, that demonstrates the effectiveness of the factored problem representations employed by SAT-based systems such as COS-P. This domain has the following important properties: (1) it has exponentially many states in the number of problem objects, (2) if there are n objects, then the branching factor is such that a breadth-first search

[6] The *de facto* winning entry at the last IPC.

encounters all the states at depth n, and (3) all plans have length n, and plan optimality is determined by the first and last actions (only) of the plan. This domain cripples state-based systems such as HSP, BASELINE, and GAMER, either because they are doing a non-factored forward heuristic search, or because —i.e., in the case of GAMER and BASELINE— they perform a breadth-first search. Finally, experiments were run on a cluster of AMD Opteron 252 2.6GHz processors, each with 2GB of RAM. All plans computed by COS-P, H-ORACLE, and BASELINE were verified by the Strathclyde Planning Group plan verifier VAL, and computed within a timeout of 30 minutes.

The results of our experiments are summarised in Table 1. For each domain there is one row for the hardest problem instance solved by each of the three planners. Here, we measure problem hardness as the time it takes each solver to return the optimal plan. In some domains we also include additional instances. Using the same experimental data as for Table 1, Figure 1 plots the cumulative number of instances solved over time by each planning system, supposing invocations of the systems on problem instances are made in parallel. It is important to note that the size of the CNF encodings required by COS-P (and H-ORACLE) are not prohibitively large – i.e, where the SAT-based approaches fail, this is typically because they exceed the 30 minutes timeout, and not because they exhaust system memory.

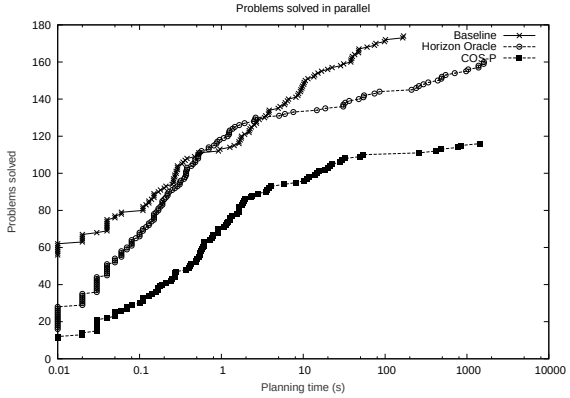


Fig. 1. The number of problems solved in parallel after a given planning time for each approach.

COS-P outperforms the BASELINE in the BLOCKS and FTB domains. For example, on BLOCKS-18 BASELINE takes 39.15 seconds while COS-P takes only 3.47 seconds. In other domains BASELINE outperforms COS-P, sometimes by several orders of magnitude. For example, on problem ZENOTRAVEL-4 BASELINE takes 0.04 seconds while COS-P takes 841.2. More importantly, we discovered that it is relatively easy to find a cost-optimal solution compared to proving its optimality. For example, on MICONIC-23 COS-P took 0.53 seconds to find the optimal plan but spent 1453 seconds proving cost-optimality. More generally, this observation is indicated by the performance of H-ORACLE.

Overall, we find that clause learning procedures in PWM-RSAT cannot exploit the presence of the *very* effective delete relaxation heuristic from Π^+. Consequently, a serious bottleneck of our approach comes from the time required to solve VARIANT-II

Table 1. C^* is the optimal cost for each problem. All times are in seconds. For BASELINE t is the solution time. For H-ORACLE, n is the horizon returned by the oracle and t is the time taken to find the lowest cost plan at n. For COS-P, t_t is the total time for all SAT instances, t_π is the total time for all SAT instances where the system was searching for a plan, while t_* is the total time for all SAT instances where the system is performing optimality proofs. '-' indicates that a solver either timed out or ran out of memory.

		BASELINE	H-ORACLE		COS-P			
Problem	C^*	t	n	t	n	t_t	t_π	t_*
blocks-17	28	39.83	28	0.59	28	3.61	3.61	0
blocks-18	26	39.15	26	0.53	26	3.47	3.47	0
blocks-23	30	-	30	4.61	30	32.11	32.11	0
blocks-25	34	-	34	3.43	34	29.49	29.49	0
depots-7	21	98.08	11	64.79	-	-	-	-
driverlog-3	12	0.11	7	0.043	7	484.8	0.08	484.7
driverlog-6	11	9.25	5	0.046	-	-	-	-
driverlog-7	13	100.9	7	1.26	-	-	-	-
elevators-2	26	0.33	3	0.01	3	14	0.01	13.99
elevators-5	55	167.9	-	-	-	-	-	-
elevators-13	59	28.59	10	378.6	-	-	-	-
freecell-4	26	47.36	-	-	-	-	-	-
ftb-17	401	38.28	17	0.08	17	0.27	0.09	0.18
ftb-30	1001	-	25	0.7	25	1.95	0.7	1.24
ftb-38	601	-	33	0.48	33	1.65	0.49	1.15
ftb-39	801	-	33	0.7	33	2.35	0.67	1.69
gripper-1	11	0	7	0.02	7	15.7	0.14	15.56
gripper-3	23	0.05	15	34.23	-	-	-	-
gripper-7	47	73.95	-	-	-	-	-	-
miconic-17	13	0	11	0.07	11	785.4	0.30	785.1
miconic-23	15	0.04	10	0.12	10	1454	0.53	1453
miconic-33	22	2.19	17	2.17	-	-	-	-
miconic-36	27	9.62	22	1754	-	-	-	-
miconic-39	28	10.61	24	484.1	-	-	-	-
pegsol-7	3	0	12	0.08	12	1.63	0.23	1.41
pegsol-9	5	0.02	15	7.07	15	416.6	12.25	404.4
pegsol-13	9	0.14	21	1025	-	-	-	-
pegsol-26	9	42.44	-	-	-	-	-	-
rovers-3	11	0.02	8	0.1	8	53.21	0.08	53.13
rovers-5	22	164.1	8	69.83	-	-	-	-
satellite-1	9	0	8	0.08	8	0.92	0.1	0.82
satellite-2	13	0.01	12	0.23	-	-	-	-
satellite-4	17	6.61	-	-	-	-	-	-
storage-7	14	0	14	0.45	14	1.16	1.16	0
storage-9	11	0.2	9	643.2	-	-	-	-
storage-13	18	3.47	18	112.1	18	262.8	262.8	0
storage-14	19	60.19	-	-	-	-	-	-
TPP-5	19	0.15	7	0.01	-	-	-	-
transport-1	54	0	5	0.02	5	0.27	0.03	0.24
transport-4	318	47.47	-	-	-	-	-	-
transport-23	630	0.92	9	1.28	-	-	-	-
zenotravel-4	8	0.04	7	1.07	7	843.7	2.47	841.2
zenotravel-6	11	8.77	7	54.35	-	-	-	-
zenotravel-7	15	5.21	8	1600	-	-	-	-

instances. On a positive note, those proofs are possible, and in domains such as BLOCKS and FTB, where the branching factor is high and useful plans long, the factored problem representations and corresponding solution procedures in the SAT-based setting payoff. Moreover, in fixed-horizon cost-optimal planning, the SAT approach continues to show good performance characteristics in many domains.

6 Concluding Remarks

In this paper we demonstrate that a general theorem-proving technique, particularly a DPLL procedure for Boolean SAT, can be modified to find cost-optimal solutions to propositional planning problems encoded as SAT.[7] In particular, we modified SAT solver RSAT2.02 to create PWM-RSAT, an effective partial weighted MaxSAT procedure for problems where all *soft* constraints are unit clauses. This forms the underlying optimisation procedure in COS-P, our cost-optimal planning system that, for successive horizon lengths, uses PWM-RSAT to establish a candidate solution at that horizon, and then to determine if that candidate is globally optimal. Each candidate is a minimal cost step-bounded plan for the problem at hand. That a candidate is globally optimal is known if no step-bounded plan with a relaxed suffix has lower cost. To achieve that, we developed a MaxSAT encoding of bounded planning problems with a relaxed suffix. This constitutes the first application of causal representations of planning in propositional logic [15].

Existing work directly related to COS-P includes the hybrid solver CO-PLAN [20] and the fixed-horizon optimal system PLAN-A. Those systems placed 4th and last respectively out of 10 systems at IPC-6. CO-PLAN is hybrid in the sense that it proceeds in two phases, each of which applies a different search technique. The first phase is SAT-based, and identifies the least costly step-optimal plan. PLAN-A also performs that computation, however assumes that a least cost step-optimal plan is globally optimal – Therefore PLAN-A was not competitive because it could not find globally optimal solutions, and thus forfeited in many domains. The first phase of CO-PLAN and the PLAN-A system can be seen as more general and efficient versions of the system described in [21]. The second phase of CO-PLAN breaks from the planning-as-SAT paradigm. It corresponds to a cost-bounded anytime best-first search. The cost bound for the second phase is provided by the first phase. Although competitive with a number of other competition entries, CO-PLAN is not competitive in IPC-6 competition benchmarks with the BASELINE – The *de facto* winning entry, a brute-force A^* in which the distance-plus-cost computation always takes the distance to be zero.

Other work related to COS-P leverages SAT modulo theory (SMT) procedures to solve problems with metric resource constraints [22]. SMT-solvers typically interleave calls to a *simplex* algorithm with the *decision steps* of a backtracking search, such as DPLL. Solvers in this category include the systems LPSAT [22], TM-LPSAT [23], and NUMREACH/SMT [1]. SMT-based planners also operate according to the BLACK-BOX scheme, posing a series of step-bounded decision problems to an SMT solver until an optimal plan is achieved. Because they are not globally optimal, existing SMT systems are not directly comparable to COS-P.

The most pressing item for future work is a technique to exploit SMT —and/or branch-and-bound procedures from weighted MaxSAT— in proving the optimality of candidate solutions that PWM-RSAT yields in bounded instances. We should also exploit recent work in using useful admissible heuristics for state-based search when evaluating whether horizon n yields an optimal solution [24].

[7] This was supposed to be possible, although in a very impractical sense (final remarks of [4]).

Acknowledgements: NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program. This work was also supported by EC FP7-IST grant 215181-CogX.

References

1. Hoffmann, J., Gomes, C.P., Selman, B., Kautz, H.A.: Sat encodings of state-space reachability problems in numeric domains. In: Proc. IJCAI. (2007)
2. Kautz, H.A.: Deconstructing planning as satisfiability. In: *Proc.* AAAI. (2006)
3. Russell, R., Holden, S.: Handling goal utility dependencies in a satisfiability framework. In: Proc. ICAPS. (2010)
4. Giunchiglia, E., Maratea, M.: Planning as satisfiability with preferences. In: Proc. ICAPS. (2007)
5. Robinson, N., Gretton, C., Pham, D.N., Sattar, A.: Sat-based parallel planning using a split representation of actions. In: Proc. ICAPS. (2009)
6. Streeter, M., Smith, S.: Using decision procedures efficiently for optimization. In: Proc. ICAPS. (2007)
7. Rintanen, J.: Evaluation strategies for planning as satisfiability. In: *Proc.* ECAI. (2004)
8. Kautz, H., Selman, B.: Unifying SAT-based and graph-based planning. In: *Proc.* IJCAI. (1999)
9. Keyder, E., Geffner, H.: Soft goals can be compiled away. Journal of Artificial Intelligence Research **36**(1) (2009)
10. Bylander, T.: The computational complexity of propositional strips planning. Artificial Intelligence **69** (1994) 165–204
11. Josep Argelic and, C.M.L., Manya, F., Planes, J.: The first and second max-sat evaluations. Journal on Satisfiability, Boolean Modeling and Computation **4** (2008) 251–278
12. Fu, Z., Malik, S.: On solving the partial max-sat problem. In: SAT 2006. (August 2006) 252–265
13. Blum, A., Furst, M.: Fast planning through planning graph analysis. Artificial Intelligence (90) (1997) 281–300
14. Rintanen, J.: Planning graphs and propositional clause learning. In: *Proc.* KR. (2008)
15. Kautz, H., McAllester, D., Selman, B.: Encoding plans in propositional logic. In: *Proc.* KR. (1996)
16. Pipatsrisawat, K., Darwiche, A.: Rsat 2.0: SAT solver description. Technical Report D–153, Automated Reasoning Group, Computer Science Department, UCLA (2007)
17. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an Efficient SAT Solver. In: Proc. DAC. (2001)
18. Marques-Silva, J.P., Sakallah, K.A.: Grasp - a new search algorithm for satisfiability. In: Proc. ICCAD. (1996)
19. Huang, J.: The effect of restarts on the efficiency of clause learning. In: Proc. IJCAI. (2007)
20. Robinson, N., Gretton, C., Pham, D.N.: Co-plan: Combining SAT-based planning with forward-search. In: Proc. IPC-6. (2008)
21. Büttner, M., Rintanen, J.: Satisfiability planning with constraints on the number of actions. In: Proc. ICAPS. (2005)
22. Wolfman, S.A., Weld, D.S.: The LPSAT engine and its application to resource planning. In: Proc. IJCAI. (1999)
23. Shin, J.A., Davis, E.: Processes and continuous change in a sat-based planner. Artif. Intell. **166**(1-2) (2005) 194–253
24. Helmert, M., Domshlak, C.: Landmarks, critical paths and abstractions: What's the difference anyway? In: Proc. ICAPS. (2009)

Robot Control Using a Switching Classical/Decision-Theoretic Planner

Richard Dearden* Charles Gretton* Michael Brenner†
Moritz Göbelbecker†

July 30, 2010

Abstract

Planning problems where the state of the world cannot be determined with certainty but must be inferred from observations are traditionally represented as partially observable Markov decision problems (POMDPs). However algorithms that operate on POMDPs are restricted to very small problems due to their computational cost. In this paper we describe an approach to plan in these domains by combining a classical planner and a POMDP planner. The approach builds plans as if the world was observable at execution time using the classical planner. These plans include assertions that state variables have particular values. When execution of the plan reaches one of these variables, the POMDP planner is called to generate a plan to determine the value of the variable. This plan can then be executed, the value determined, and the classical plan can then continue executing if the value determined was the one desired, or if the variable was found to have a different value, replanning occurs. We show how this approach can generate plans in domains that are much too large for POMDP planners to solve directly.

1 Introduction

Real-world robotic planning contains many sources of uncertainty, including unpredictable action effects (particularly for actions such as dialogue), unknown world state, and the lack of a closed world assumption. In typical hierarchical robotic architectures for planning, such as 3T [1], this uncertainty is removed as much as possible from the planner's consideration by hand-building a set of low-level reactive control policies. For example, a high-level action "goto-door" might be implemented with a control policy that drives the robot forwards a small distance at a time until the door is reached. This approach works particularly well for removing uncertainty in action effects.

For state uncertainly, a similar approach may be taken in which, when the high-level planner needs to know the value of a particular state variable, a lower-level routine

*School of Computer Science, University of Birmingham, UK, {rwd,grettonc}@cs.bham.ac.uk
†Albert Ludwig University,

attempts to discover the value. In this paper we explore how such a routine can be generated automatically when needed.

An alternative approach to planning with uncertainty in state and action effects is to use a decision-theoretic planning approach such as a POMDP solver [6, 2, 4]. For quite specialised tasks such as observation planning (see [7] for example) this approach can be very successful. However, for more realistic problems the state space is simply too large for the state of the art in POMDP solvers.

Our approach is to build a planner that switches between classical and decision-theoretic planning depending on the situation. The idea is to represent the problem in a rich enough language that a POMDP solver could solve the problem, but to derive a classical planning problem from that representation. We then construct a plan for the classical planning problem which may involve actions that observe the value of particular state variables. Each of these observation actions corresponds to an invocation of the POMDP planner but on a much reduced state space from the original problem.

The classical planner we use is Continual Planning [3], which generates a straight line plan for the original problem but may replan if the world doesn't match the assumptions it made in constructing that plan. One can think of this as contingency planning where only a single branch is planned for, and if at execution time a different branch should be followed, replanning is triggered to generate that branch. Most of the replanning points occur when a state variable is observed. Thus the overall flow of execution typically consists of executing a fragment of a classical plan, followed by some decision-theoretic plan to observe a variable. After this, some replanning may occur, and then the process repeats until the goal is reached.

In the following section we briefly present the Continual Planner we use for generating classical plans. After that we briefly describe the POMDP planner we use. In Section 4 we describe the original input language and how that gets translated into a classical planning language. Finally, we put everything together in Section 5 and present some variations on the algorithm and difficulties with the current approach in Sections 6 and 7.

2 Continual Planning

Instead of considering many possible futures in advance, an agent may execute parts of its plan in order to gather additional information, thereby reducing the number of possible contingencies that it has to take into account for the remaining planning. This technique of interleaving planning, plan execution and execution monitoring is called Continual Planning (CP). Our CP approach is based on the idea of active knowledge-gathering: instead of planning for possible contingencies, agents try to learn more about the state of the world directly. In order to enable agents to reason about how they can gather additional knowledge it is necessary to explicitly model the agents' beliefs as well as their sensing capabilities as part of their formal planning domain. Agents can then plan how to extend their knowledge. In contrast to Contingent Planning approaches which have used similar ideas, we do not want the planning process to branch over the possible outcomes of sensing actions. Instead we want to enable a planner to postpone the decision of what exactly to do with that knowledge to the moment where

the actual perception has been made. In other words, we want to "hide" a conditional subplan until the agent has enough information to plan its details. For this purpose, we introduced the concept of assertions. Assertions are specific actions, defined normally in the planning domain. However, assertions cannot be directly executed. Instead, the domain designer asserts that the effects of the assertion can be achieved by a subplan if its preconditions are satisfied.

3 The Decision-Theoretic Planner

The current *de facto* model for probabilistic decision-theoretic planning with partial observability is the POMDP. For our purposes, a POMDP is a six-tuple $\langle \mathcal{S}, \mathcal{A}, \mathsf{Pr}, \mathsf{R}, \mathbb{O}, v \rangle$. Here, $\mathcal{S}$, $\mathcal{A}$, Pr, and R are states, actions, state-transition function, and reward function, respectively – They provide an MDP-based specification of the underlying world state, dynamics, and reward. $\mathbb{O}$ is a set of observations. For each $s \in \mathcal{S}$ and action $a \in \mathcal{A}$, an observation $o \in \mathbb{O}$ is generated independently according to some probability distribution $v(s, a)$. We denote $v_o(s, a)$ the probability of getting observation o in state s. For s and a we have the following constraint:

$$\sum_{o \in \mathbb{O}} v_o(s, a) = 1$$

The optimal solution to a finite-horizon POMDP problem can be expressed as a policy $\mu : \mathbb{O}^* \rightarrow P_{\mathcal{A}}$ where $\mu_a(o_0, .., o_t)$ is the probability that we execute action a given observation history $o_0, .., o_t$.[1] In practice for the problems we are interested in solving with the decision-theoretic planner the policies are quite short and so we generate them using an implementation of LAO* [5] modified to work on POMDPs.

Hansen and Zilberstein's LAO* algorithm is a generalisation of AO* that can be used to solve Markov decision problems (MDPs) where the start state is given, but is particularly suited to stochastic shortest path problems—MDPs with absorbing goal states where the object is to reach the goal state with the lowest expected cost. Figure 1 is a high-level description of the LAO* algorithm. Essentially it works by alternating between expanding the current search graph (Steps 1–5) and performing dynamic programming to determine which actions to perform in that graph (Steps 6–11). Note that the value iteration in Step 7 may result in G^+ changing (because the best action for a state changes), so at each iteration the set of states on which value iteration is performed may change.

The advantage of LAO* is that it rarely adds all the states in the MDP to the explicit graph, and adds even fewer of them to G^+. This means that with a good heuristic it can solve extremely large MDPs. As we will see in the next section, this is potentially important in our case as the state space is the cross product of the belief states reachable under the low-level policy for each ROI.

[1] Such a policy can oftentimes be compactly represented as a tree or algebraic decision diagram (ADD).

Require: G is the explicit graph (the current search graph), initially containing only the start state. G^+ is the subset of G that currently looks best.
Require: $h(s)$ is a heuristic for the state values
Require: $goal(s)$ is true if s is a goal state

1: **while** there are unexpanded nodes in G^+ **do**
2: Expand one of the unexpanded nodes s in G^+ by adding all its successors s' to G.
3: $\forall s'$, if $goal(s')$ then $f(s') = 0$ else $f(s') = h(s')$.
4: Run value iteration on s and all its ancestors in G^+.
5: **end while**
6: Perform value iteration on G^+.
7: **if** the error between successive iterations of value iteration falls below ϵ **then**
8: Return G^*
9: **else**
10: Goto Step 1.
11: **end if**

Figure 1: The LAO* algorithm.

4 Input Language

The complete switching planner system uses a variant of the standard PPDDL1.0 planning language [8] that supports decision-theoretic problem definitions, which we call decision-theoretic PDDL (DTPDDL). For lack of space we will leave out the complete language definition and give a small example from a simplified version of the Dora domain here.

```
(define (domain cogx)

        (:requirements :mapl :adl :object-fluents :partial-observability)

        (:types
             place room - object
             robot - agent
             robot - movable
             place_label - object
        )

        (:constants
             kitchen office library - place_label
        )

        (:predicates
             (connected ?n1 - place ?n2 - place)
             (visited ?r - robot ?p - place)
        )

        (:functions
             (label ?p - place) - place_label
             (is-in ?r - movable) - place
        )

        (:action sense-placelabel
             :agent (?a - robot)
             :parameters (?loc - place)
             :precondition (= (is-in ?a) ?loc)
             :effect ()
        )
```

```
    (:observe placelabel
         :agent (?a - robot)
         :parameters (?loc - place ?n - place_label)
         :execution (sense-placelabel ?a ?loc)
         :effect (and (when (= (label ?loc) ?n)
                            (probabilistic 0.7 (observed (label ?loc) ?n)))
                      (when (not (= (label ?loc) ?n))
                            (probabilistic 0.1 (observed (label ?loc) ?n))))
    )

    (:action move
         :agent (?a - robot)
         :parameters (?to - place)
         :variables (?from - place)
         :precondition (and
                        (= (is-in ?a) ?from)
                        (or (connected ?from ?to)
                            (connected ?to ?from)))
         :effect (assign (is-in ?a) ?to)
    )

)
```

Here, we have only two actions, a MOVE action that gets the robot from place to place and a SENSE-PLACELABEL action that makes observations of the room the robot is in. These observations are modelled using the PLACELABEL observation which states that 70 percent of the time the true label of the room will be returned, but 10 percent of the time an incorrect label will be returned.

In addition to the domain definition we also need to define a planning problem as follows:

```
(define (problem cogxtask)

  (:domain cogx)

  (:objects  robot_0 - robot
             place_0 place_1 place_2 - place
  )

  (:init  (connected place_0 place_1)
          (connected place_1 place_0)
          (connected place_1 place_2)
          (connected place_2 place_1)
          (= (is-in robot_0) place_0)
  )

  (:goal  (forall (?p - place)  (kval robot_0 (label ?p))))
)
```

Here we define the objects in the world, the initial state (the connections between rooms and the location of the robot) and the goal, which in this case is to know the label for all the places in the world.

5 The Switching Planner

As we said above, in typical robotic systems, decision-making is accomplished by a three level architecture such as 3T [1] with a classical planner at the highest level, simple control sequences at the lowest level and a middle level that translates the high-level actions into alternative sequences of controllers. The reason such an architecture

is favoured is that the high-level planner typically cannot cope with uncertainty in the world, so the rest of the architecture is designed to remove as much of that uncertainty as possible. Typically it is assumed that this uncertainty is in the form of *uncertain action effects*.

In our domain we find ourselves with a much harder kind of uncertainty to handle, namely *uncertainty in the state* of the world. The Dora domain involves trying to find objects that are in unknown locations and trying to explore the world and learn more about it. These problems are inherently ones where the robot has only partial information about the state of the world at any time. The actions that are available to us from the rest of the robotic system (for example, driving actions) of course contain the same kinds of low-level control as would be found in a 3T system, but these aren't generally useful for removing state uncertainty. Instead, what we need are information gathering actions such as visual search and dialogue that allow us to reduce our uncertainty about the state variables we care about.

The intuition behind our switching planner is to think of the actions provided by the rest of the system as the low-level actions in a three level robotic architecture, to use a classical planner (continual planning) at the high-level as normal, but to automatically generate the middle layer of the architecture *on the fly* using the decision-theoretic planner.

To do this, we require a representation of the actions given to the overall planning system that is rich enough to represent all the details we need for this middle layer. That means that the representation must contain observations of the world state, and those observations can be unreliable. The decision-theoretic planning domain definition language (DTPDDL) presented in Section 4 provides a common language for both planners from which classical planning actions suitable for CP can be extracted, but which can also be used by the decision-theoretic planner.

The overall architecture of the switching planner is given in Figure 2. The input is a problem specification in DTPDDL made up of three parts:

- A domain definition that specifies the actions, which at present is pre-built, but which in the future will be learned from experience.
- An initial state that comes from working memory (specifically the binder).
- A goal or goals, which come from the motivation subarchitecture.

The planning problem is translated into a form that CP can plan with, and CP generates a plan. This plan contains no branching points, but instead it contains points where domain variables are observed and a particular value is assumed. Thus we can think of it as a branching plan—with branches that depend on the values of domain variables at execution time—where only one branch at each branch point has actually been planned for. If a different branch is needed during execution, then the planner may be called again to replan for that branch.

During execution of the CP plan, when a domain variable is observed, the plan executor needs a way to determine the value of that variable. When the value isn't known with certainty, then the decision-theoretic planner is called to determine the value. It is given the same DTPDDL domain definition and the current initial state,

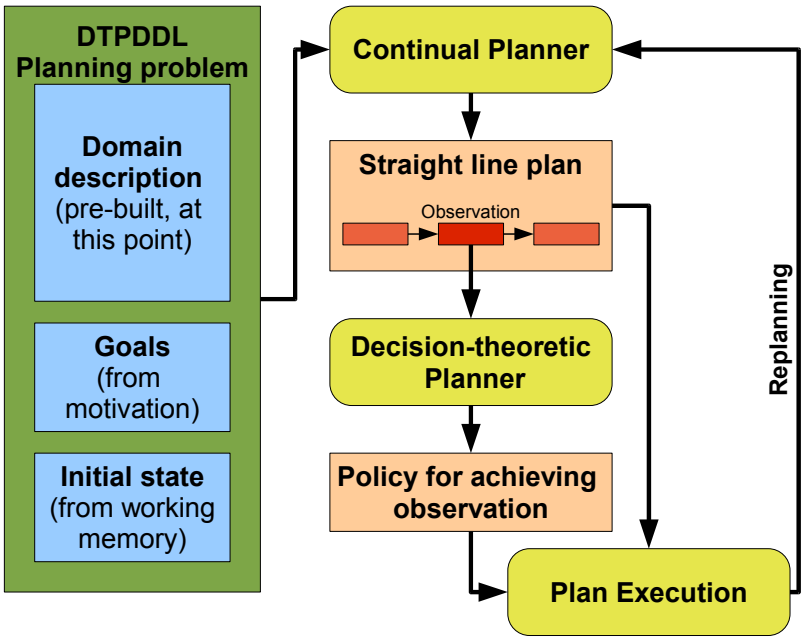


Figure 2: The architecture of the switching planner.

but the goal is to determine with sufficient confidence the value of the variable. The decision-theoretic planner produces a plan for this very small problem, which can then be executed to find out the value. At that point, the plan executor can either continue with the CP plan, or if necessary trigger CP to replan.

As an example, consider the problem definition we gave above. Here the problem was to determine the label on each of the three rooms. The DTPDDL problem is translated into a deterministic problem domain that CP can plan with as shown below:

```
(define (domain cogx)

(:requirements :mapl :adl :object-fluents)

(:types  place room - object
         robot - agent
         robot - movable
         place_label - object
)

(:predicates  (connected ?n1 - place ?n2 - place)
)

(:functions  (is-in ?r - movable) - place
             (label ?p - place) - place_label
)

(:constants  kitchen library office - place_label
)

)
```

```
(:action  sense-placelabel
          :agent  (?a - robot)
          :parameters  (?loc - place)
          :precondition  (= (is-in ?a) ?loc)
          :effect
          :sense  (label ?loc) ;; == (kval ?a (label ?loc))
)

(:action  move
          :agent  (?a - robot)
          :parameters  (?to - place)
          :variables  (?from - place)
          :precondition  (and  (= (is-in ?a) ?from)
                               (or  (connected ?from ?to)
                                    (connected ?to ?from)
                               )
                         )
          :effect  (assign (is-in ?a) ?to)
)

)
```

CP then plans for this problem (with the same initial state), and produces the following plan:

```
(sense-placelabel robot_0 place_0)
(move robot_0 place_1 place_0)
(sense-placelabel robot_0 place_1)
(move robot_0 place_2 place_1)
(sense-placelabel robot_0 place_2)
```

Each of the SENSE-PLACELABEL actions is an observation of the label of a room, and triggers a call to the decision-theoretic planner with the original DTPDDL domain but with the following action added:

```
(:action  commit-label-place_0
          :parameters  (?val - place_label)
          :precondition  (not (committed (label place_0)))
          :effect  (and  (committed (label place_0))
                         (when  (= (label place_0) ?val)
                                (increase (reward ) 100)
                         (when  (not {= (label place_0) ?val))
                                (decrease (reward ) 100)
                         )
                   )
)
```

This extra action is specific to the first execution of SENSE-PLACELABEL, and allows the decision-theoretic planner to accumulate reward by committing to the room having the specified label.

The decision-theoretic planner now generates a plan for this domain, which consists of multiple executions of SENSE-PLACELABEL until the likelihood according to the belief state of one of the labels is sufficiently high that the reward is maximised. This plan can then be executed and CP will then move on to the next step in the plan. In this case, since the robot doesn't have anything to do depending on the label of each room, there are no branch points in the CP plan, so no replanning occurs.

6 Variations

The version of the switching planner we have described above only triggers the decision-theoretic planner when execution reaches a point at which a variable is observed. This is efficient in the sense that only observations that are actually executed get planned for, but it is inefficient in that the policy for determining the value of the variable is only generated once execution reaches that point. An alternative is to perform some or all of the decision-theoretic planning in advance. This has the advantage that the computation is being performed while the CP plan is executing. If the CP plan contains move actions for example, then a significant amount of planning can be accomplished with no time penalty. However, this approach requires the state on reaching the observation in the CP plan to be determined. This can be done by simulating the CP plan using the full DTPDDL representation of the actions, but if those actions are non-deterministic, there may be several possible initial states for the decision-theoretic planner. If all of these are planned for, the cost of planning may be very high.

These issues produce an interesting space of possible pre-planning strategies which we are in the process of exploring. The first option is whether to pre-plan the decision-theoretic problems in advance, and if so, how many of them to plan. The second is whether to plan for every possible initial state, or only some subset chosen on the basis of their likelihood. All of this is of course dependent on the amount of planning time available.

7 Disadvantages of this Approach

The approach we describe is necessary in that somehow we need to determine the values of the variables CP plans to observe. However, it has a number of disadvantages.

The first disadvantage is that because CP has no concept of probability, the plans it builds are designed to be short, rather than being likely to succeed without replanning. Thus, if CP is asked to plan to find the kitchen, it will build a plan that goes to the nearest room with an unknown label, observe that it is a kitchen, and finish. If we already have a lot of evidence that the room is not a kitchen, we have no way of telling CP that perhaps going to a different room would be better.

The second problem is that it may be impossible to observe the value of a variable CP asks for. At present we force the decision-theoretic planner to commit to the most likely value for a variable after a fixed number of steps (that is, it can only build n-step plans). This may lead to errors where the planner thinks a variable has a particular value while working memory records the value as still uncertain. We are investigating ways to overcome this such as allowing the decision-theoretic planner to return UNKNOWN as its answer, which would then trigger replanning in CP. However, it is hard in this circumstance to stop CP finding exactly the same plan again.

A final issue is that we cannot evaluate the quality of the plans we find. This is because CP doesn't actually generate the whole plan, only a single branch of it. If we had access to the complete branching plan, we could start to think about plan quality over the whole plan, not just the decision-theoretic part. We are planning to investigate this issue in more depth in Year 3.

The ideal solution to all these problems would be to run the decision-theoretic planner on the original planner. However, as we pointed out above, the whole point of this approach is to keep computation times manageable. We simply can't produce acceptable real-time performance with the decision-theoretic planner, so the problems we describe here are the trade-off we have to make for acceptable performance. One possibility we will investigate in Year 3 is to use the decision-theoretic planner on other parts of the plan where plan quality will make a significant difference. However, in general, working out which of the parts of the problem would benefit from a decision-theoretic solution is as hard as solving the planning problem with the decision-theoretic planner.

Acknowledgements

This research was supported by the EU as part of the Integrated Project CogX (FP7-ICT-2xo15181-CogX).

References

[1] R. P. Bonasso, D. Kortenkamp, and T Whitney. Using a robot control architecture to automate space shuttle operations. In *Proceedings of IAAI*, 1997.

[2] Xavier Boyen and Daphne Koller. Tractable inference for complex stochastic processes. In *UAI*, pages 33–42. AUAI Press, 1998.

[3] M. Brenner and B. Nebel. Continual planning and acting in dynamic multiagent environments. *Journal of Autonomous Agents and Multiagent Systems*, 2009. to appear.

[4] Zhengzhu Feng and Eric Hansen. Approximate planning for factored pomdps. In *In Proceedings of the Sixth European Conference on Planning*. Springer, 2001.

[5] Eric Hansen and Schlomo Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129, 2001.

[6] E. J. Sondik. *The Optimal Control of Partially Observable Markov Decision Processes*. PhD thesis, Stanford, California, 1971.

[7] Mohan Sridharan, Jeremy Wyatt, and Richard Dearden. Planning to see: Hierarchical POMDPs for planning visual actions on a robot. *To appear in Artificial Intelligence Journal*, 2010.

[8] H. L. S. Younes and M. Littman. PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2004.