22nd International Conference
on Automated Planning and Scheduling

June 25-29, 2012, Atibaia – Sao Paulo – Brazil

PlanEx 2012

Proceedings of the Workshop on Planning and Plan Execution for Real-World Systems: Principles and Practices

Edited by
Frédéric Py, David Musliner

Organization

Frédéric P. Py, MBARI, United States
contact email: fpy@mbari.org

David J. Musliner, SIFT LLC, United States
contact email: musliner@sift.net

Program Committee

Mark Boddy, Adventium, United States

Bradley Clement, JPL, United States

Paul Morris, NASA Ames Research Center, United States

Nicola Muscettola, Google, United States

Andrea Orlandini, IP-CNR, Italy

Kanna Rajan, MBARI, United States

Alessandro Saffiotti, Orebro University, Sweden

Florent Teichteil, ONERA-CERT, France

Gérard Verfaillie, ONERA-CERT, France

Foreword

Early approaches to robot control were based on the Sense-Plan-Act (SPA) paradigm with planning as the core of a control-loop. Using this paradigm, real-world applications merging deliberative and reactive decision have made remarkable strides in the last few years. These systems have evolved from the classical concept of three-layered control running off-board, to demonstrate dynamic control of a multitude of platforms using onboard and hybrid mixed-initiative techniques. In the meantime, automated planning techniques have evolved substantially in the areas of modeling, reasoning methods, and search algorithms. Together these advances open up new possibilities for how planning technology can be applied in execution, but also reveal new concerns like the interaction between different decisional components or the possible conflict between decision and environmental reality. The goal of this workshop is to integrate practical experience in fielded autonomous systems with theoretical and empirical results in automated planning to stimulate new perspectives on the roles and requirements for planning in execution.

Frédéric Py, David Muslinner
PlanEx 2012 Organizers
June 2012

Contents

Variable resolution planning through predicate relaxation

Moisés Martínez
Universidad Carlos III de Madrid
Avenida de la Universidad, 30
28911 Leganés, Spain
moises.martinez@uc3m.es

Fernándo Fernández
University Carlos III de Madrid
Avenida de la University, 30
28911 Leganés, Spain
fernando.fernandez@uc3m.es

Daniel Borrajo
University Carlos III de Madrid
Avenida de la University, 30
28911 Leganés, Spain
dborrajo@ia.uc3m.es

Abstract

Planning a sequence of actions is particularly difficult in stochastic environments, where actions execution might fail, which in turn prevents the execution of the rest of the plan. Also, in many domains, agents cannot wait a long time for plan generation. In this paper, we propose the use of Variable Resolution Planning (VRP). The main idea is based on the fact that if the domain is stochastic it is usually not worth computing a valid (sound) long plan, since most of it will not really be used. So, VRP generates a plan where the first k actions are applicable if the environment does not change, while the rest of the plan might not be necessarily applicable, since it has been generated using an abstraction by removing some predicates. Also, planning with abstraction requires less computation time than computing regular applicable plans. Experimental results show the advantages of this approach in several domains over a standard planning approach.

Introduction

Stochastic environments are challenging for Automated Planning, where a planner has to generate a plan of actions, and the plan execution may yield unexpected states. This process can be prohibitively expensive for most real world scenarios. Besides, depending on the stochasticity level of the domain, most of the actions of the generated plan will not be even applied when the execution of the previous actions in the plan generate an unexpected state. In the worst case an unexpected state can be a failure state from which the rest of the plan cannot be successfully executed.

There are different ways to approach planning and execution in stochastic domains. Different techniques can be applied depending of the information known about the environment. If we have information about the dynamics of the environment (failure in the actuators of a robot, the structure of the terrain, accuracy of sensors, etc), we can define a domain model with probabilistic information (such as in PPDDL). Then, we can build a conditional plan (Peot and Smith 1992) where the plan takes into account all possible alternative problems, or generate a set of policies by solving it as a Markov Decision Process (MDP) as shown by LAO* (Hansen and Zilberstein 2001), or LRTDP (Bonet and Geffner 2004). But, usually, the dynamics of the environment is not known. Then, in turn we have two alternatives. First, we can learn the dynamics. However, usually the learning effort is impractical except for small problems (Zettlemoyer, Pasula, and Kaelbling 2005). The second solution, and most used one, consists of using a deterministic domain model and replan when a failure in execution is detected.

A common solution is using a repair or re-planning system (Fox et al. 2006; Yoon, Fern, and Givan 2007; Borrajo and Veloso 2012). In re-planning, the planner generates an initial applicable plan and executes it, one action at a time. If an unexpected state is detected, the system generates a new plan. This process is repeated until the system reaches the problem goals. Therefore, at each planning (re-planning) step, including the initial one, the system is devoting a huge computational effort on computing a valid plan (applicable plan that achieves the goals), when most of it will not be applied. On the other extreme, reactive systems require much less computational effort. They greedily select the next action to be applied according to some configured - or learned - knowledge. They are "mostly" blind with respect to the future; they usually ignore the impact of the selected action on the next actions and states. Thus, they often get trapped in local minima, or dead-ends.

We advocate here for an approach in which a deterministic planner devotes less time to compute a valid first portion of the plan, and checks that there is a potential continuation of the plan by means of abstractions, trying to avoid dead-ends and/or decreasing computational effort. One of the advantages of our approach is that it requires less planning time than traditional planning approaches that compute a valid complete plan, while retaining their capability of reasoning into the future. As explained before, we assume lack of information about the dynamics of the environment, so we use a deterministic STRIPS domain model and replan upon failure.

To design this approach, we have been inspired by the work of Zickler and Veloso (Zickler and Veloso 2010), where a motion planning technique is presented. It generates a collision-free trajectory from an initial state to a goal state in dynamic environments. They introduced the concept of Variable Level-Of-Detail (VLOD), which focuses search on obtaining accurate short-term motion planning, while con-

sidering the far future with a different level of detail, by selectively ignoring the physical interactions with dynamic objects. This technique decreases the computational cost of the motion planning process, so that information about different elements of the environment is not used to search a path to reach all goals. We introduce Variable Resolution Planning (VRP) through predicate relaxation, a new approach to reduce the computational overhead of planning in stochastic environments. It is based on two ideas: some effort is devoted to compute a valid head of the plan of length k; and the rest of the plan is checked for potential reachability by relaxing the complexity of actions and decreasing domain details through abstraction. Our approach is based on selectively ignoring some domain details, in our case predicates.

This paper is organized as follows: first in Section 2, we formally define the representation of the planning problem. In Section 3 we define our abstraction mechanism. In Section 4 we describe how to implement Variable Resolution Planning (VPR). In Section 5 we show an experimental evaluation of our techniques over different domains. Section 6 presents some work related with our approach. Finally, in Section 7 we conclude and introduce future work.

Planning Framework

In this work, we focus on the use of a classical Automated Planning approach under stochastic environments. A classical planning problem is defined in PDDL using a lifted representation in predicate logic, but most current planners always perform first a grounding transformation. Under that transformation, a planning problem can be defined as a tuple $P = (F, A, I, G)$, where:

- F is a finite set of relevant grounded literals (also known as facts), functions and fluents.
- A is a finite set of grounded actions derived from the action schemes of the domain, where each action $a_i \in A$ can be defined as a tuple $a_i = (Pre, Add, Del)$, where $Pre(a_i), Add(a_i), Del(a_i) \subseteq F$, $Pre(a_i)$ are the preconditions of the action, $Add(a_i)$ are its add effects, and $Del(a_i)$ are the delete effects. $\mathit{Eff}(a_i) = Add(a_i) \cup Del(a_i)$ are the effects of the action.
- $I \subseteq F$ is the initial state.
- $G \subseteq F$ is a set of goals.

A plan π for a problem P is a set of actions (in the common case a sequence) $\pi = (a_1, \ldots, a_n), \forall a_i \in A$, that transforms the initial state I into a final state S_n where $G \subseteq S_n$. This plan π can be executed if the preconditions of each action are satisfied in the state in which it is applied, i.e. $\forall a_i \in \pi\, Pre(a_i) \subseteq S_{i-1}$ ($S_0 = I$).

Abstractions

An abstraction can be defined as a function that transforms a planning problem into another (simpler) one, where low-level details are ignored. Usually, abstractions help on reducing the problem complexity. In this work, we generate abstractions by removing some predicates from the lifted representation of the domain. Thus, since planners work at the propositional (instantiated) level, it is equivalent to remove literals from the preconditions and effects of actions in the grounded representation. We will now define some concepts related to our abstraction mechanism.

The first definition defines a mapping between a predicate in the PDDL domain definition and its corresponding groundings for problem P.

Definition 1 *$g(p, P)$ is the set of propositions (facts) of predicate p in problem P.*

For instance, in the Blocksworld domain, if p is `(ontable ?x)` and a PDDL problem P_1 defines three blocks (A, B, and C) that are on the table, then:

$$g(p, P_1) = \{\texttt{(ontable A), (ontable B), (ontable C)}\}$$

In this approach, an abstraction of an action a over a predicate p removes all propositions that are groundings of predicate p – $g(p, P)$ – from the preconditions and effects of action a. In order to select which predicates to remove, we split the set of facts F into three subsets: $F_d \subseteq F$ is the set of dynamic facts, or facts that appear in the effects of at least one action ($F_d = \{f \in F \mid \exists a_i \in A, f \in \mathit{Eff}(a_i)\}$). $F_s \subseteq F$ is the set of static facts, or facts that do not appear in the effects of any action; and $F_f \subseteq F$ is the functions set, or non-boolean facts that are groundings of PDDL functions instead of PDDL predicates. We are not going to remove all domain predicates (or their corresponding facts), so we define the candidate subset of facts to be potentially removed, $C^{abs} \subseteq F$. We will not use all predicates to generate abstractions. First, static predicates are not added or deleted during the planning process. Therefore, applying an abstraction over a static predicate does not reduce the number of actions to achieve the goals. Deleting an static predicate only increases the number of actions that may be selected during the search process to reach the goal state. Second, predicates which are part of the goal subset cannot be removed, because the planner would not reach a solution. And, finally, we are only

considering STRIPS domains in this paper. C^{abs} is composed of all facts except for the static facts (F_s), the functions facts (F_f) and facts that are part of the problem goals (G). Thus:

$$C^{abs} = F \setminus (F_s \cup G \cup F_f)$$

Next, we include our definitions of abstractions:

Definition 2 *An abstraction of an instantiated action $a \in A$ over a predicate p in problem P is defined by function $f(a, p) = (Pre_p^{abs}(a), Add_p^{abs}(a), Del_p^{abs}(a))$, where:*

- $Pre_p^{abs}(a) = Pre(a) \setminus g(p, P)$
- $Add_p^{abs}(a) = Add(a) \setminus g(p, P)$
- $Del_p^{abs}(a) = Del(a) \setminus g(p, P)$.

For instance, in the Blocksworld domain, given a problem P_1 previously defined, if a_1 is `pick-up(A)` and predicate p_1 is `(ontable ?x)`. If we select to remove p_1, $g(p_1, P_1) = \langle (ontableA), (ontableB), (ontableC) \rangle$.

- $Pre_p^{abs}(a_1) = \langle(clearA)(ontableA)(handempty)\rangle \setminus \langle(ontableA),(ontableB),(ontableC)\rangle$
- $Add_p^{abs}(a_1) = \langle(holdingA)\rangle \setminus \langle(ontableA),(ontableB),(ontableC)\rangle$
- $Del_p^{abs}(a_1) = \langle(ontableA),(clearA),(handempty)\rangle \setminus \langle(ontableA),(ontableB),(ontableC)\rangle$.

Definition 3 *An abstraction of an instantiated action a over a set of predicates $L = \{p_1, \ldots, p_n\}$ in problem P is the result of iteratively abstracting action a over each predicate $p_i \in L$. The process can be recursively defined as:*

- $f(a, \{p\}) = f(a, p)$
- $f(a, \{p_1, \ldots, p_n\}) = f(f(a, \{p_2, \ldots, p_n\}), p_1)$

L will always be composed of PDDL predicates (in the lifted representation) whose corresponding facts (groundings) belong to C^{abs}; i.e. candidate elements of L will be in the set of candidate predicates $CL = \{p \mid g(p, P) \subseteq C^{abs}\}$.

Definition 4 *An abstraction of a PDDL problem P over a set of predicates $L \subseteq CL$, called P_L^{abs}, is the result of abstracting all components of P over L:*

$$P_L^{abs} = (F_L^{abs}, A_L^{abs}, I_L^{abs}, G_L^{abs})$$

where

- $F_L^{abs} = F \setminus \cup_{p \in L} g(p, P)$
- $A_L^{abs} = \{a^{abs} \mid a \in A, a^{abs} = f(a, L)\}$
- $I_L^{abs} = I \setminus \cup_{p \in L} g(p, P)$
- $G_L^{abs} = G$, *given that the predicates in the goals are not candidates to abstract*

Definition 5 *An abstract plan Π_L^{abs} that solves a PDDL problem P is an action sequence $\Pi_L^{abs} = \{a_0, \ldots, a_{k-1}, a_k, \ldots a_{n-1}\}$. The k first actions are applicable in P if the actions $a_i, i = 0 \ldots k$, are all in A, and there exists a sequence of states $(s_0, \ldots, s_k)$, such that $s_0 = I$, $Pre(a_i) \subseteq s_i$ and $s_{i+1} = s_i \cup Add(a_i) \setminus Del(a_i)$ for each $i = 0, \ldots, k-1$. The rest of actions from a_k are applicable in an abstract P_L^{abs} if the actions $a_i, i = k \ldots n$, are all in A_L^{abs}, and there exists a sequence of states $(s_k, \ldots, s_n)$, such that $Pre^{abs}(a_i) \subseteq s_i$ and $s_{i+1} = s_i \cup Add^{abs}(a_i) \setminus Del^{abs}(a_i)$ for each $i = k, \ldots, n$. Thus, a partial abstract plan is composed of a standard sound plan from the initial state s_0 to state s_k and an abstract plan from state s_k to a goal state.*

Variable Resolution Planning

The goal of VRP in stochastic domains is to quickly come up with a plan whose first k actions are sound and whose rest of actions could potentially solve the problem by adding the removed details. Thus, we will generate a search tree where nodes of depth less or equal to k use the original definition of the problem, and nodes below that depth will use an abstracted version of the problem. In order to use current planners under this approach, modifications in the planners code are necessary. In this work, Metric-FF (Hoffmann 2003) has been used as a base to implement our approach. We call the new planner Abstract-MFF. Apart from the standard inputs of Metric-FF, Abstract-MFF also receives k and L as input. Once the initial instantiation is performed (computing the instantiated problem and all its components, $P = (F, A, I, G)$), Abstract-MFF generates data structures to use horizon h and abstractions. It creates two different spaces for instantiated actions. The first space S is composed of the actions that can be executed for the standard problem P. The second space S^{abs} is composed of the actions that can be executed for the abstracted problem P^{abs}. Next, the planner starts the search process using ωA* (ω=5), when the depth of a node is greater than the Temporal Horizon, k, the abstract space, P^{abs}, is activated for all the nodes in the subtree of that node while the original space, P is deactivated. The implementation basically changes the space corresponding to the original problem P for the one corresponding to the abstracted problem P^{abs}. So, the nodes above depth k use S and the ones below that threshold use S^{abs}. Finally, the planner generates an abstract plan.

We used, without success, alternative ways of performing this implementation through compiling the dynamic change from the original to the abstract space into the PDDL domain model and searching on that space. For space reasons, we cannot expand here on why it did not work, but it had to do with generating bigger problem spaces (complexity), or representation capabilities of current planners (expressivity).

Experimentation

To evaluate the approach presented in this paper, we have performed a planning-execution-replanning loop on some domains and problems. We have used MDPSim for simulation of plans execution.[1] We provide it with a probabilistic version of each domain in PPDDL (Younes and Littman 2004), where actions can generate unexpected states with different probabilities. In our case an state is considered as an unexpected state when an action could not be executed in the environment and the execution returns the previous state. We have chosen the most simple case to define failures, because we did not have access to a simulator that is able to introduce exogenous effects and MDPSim does not offer a way to include them. For instance, in the Rovers domain when a robot moves from a waypoint1 to a waypoint3, three possibilities can be expected in the real world. First, the robot moves to the right waypoint (the one on the effects of the deterministic version of the actions). Second, the robot does not move, so it stays in the same waypoint. Finally, it can move to another waypoint close to the origin or destination ones.

The overall system works as shown in the algorithm in 1. Given a planning task (consisting of a deterministic PDDL domain and problem descriptions), L (the set of predicates to be removed) and k (the temporal horizon), the planner generates an abstract plan (where the first k actions are non-abstracted actions and the rest are abstracted actions). Next, it sends every action to MDPSim. MDPSim executes each action (with the stochastic model of the PDDL domain). If the resulting state is not the expected one according to the

[1] It was developed for the First Probabilistic Planning Competition (Younes et al. 2005).

Algorithm 1: Description of the execution process in the stochastic environment.

Data: Problem: $\Pi = (F, A, I, G)$, predicates L, horizon h

begin

$plan = planning(\Pi, L, h)$

$lastState = I$

while *plan is empty at the current state do* **do**

$action \longleftarrow getAction(plan)$

$expected \longleftarrow$ $generateState(action, lastState)$

$state \longleftarrow sendActionToMDPSIM(action)$

if $state <> excepted$ **then**

$\Pi \longleftarrow$ $generatePlanningProblem(\Pi, lastState)$

$plan = planning(\Pi, L, h)$

else

$lastState \longleftarrow state$

end

deterministic version of the domain, we generate a new plan from that state. This process is repeated as a re-planning loop until a goal state is reached. We compare Abstract-MFF with the standard Metric-FF planner.

Due to the characteristics of the simulator, we assume that the different elements (robots, trucks, satellites) can wait in a secure state during the computation or execution of a plan.

Experiments have been done on an Intel Xeon 2.93 GHZ Quad Core processor (64 bits) running under Linux. The maximum available memory for the planners was set to 6 GB, the maximum planning time for a problem has been set to 1000 seconds and the maximum execution time has been set to 30000 seconds. Each problem has been executed fifteen times until goals are reached (we call it a run). The following provides an evaluation of this approach over four benchmark domains: Rovers, DriverLog, Depots and Satellite domains from IPC-3. First, we present a deep analysis on the Rovers domain to study the different effects of this technique changing several characteristics. And second we show the results on the rest of domains.

Rovers Domain

The technique presented in this paper has been designed to be applied in stochastic environments. For this reason, the initial testing of this technique has been performed on a domain generated for a real world problem, the Rovers Domain. It was designed for the sequential track of IPC-3 (2002) and was inspired on the Mars exploration rovers missions where an area of the planet is represented as a grid of cells, called waypoints. They contain samples of rock or soil that can be collected by the robots. Each robot can traverse across different waypoints and can perform a set of different actions (analyze rock or soil samples or take pictures of a specific waypoint). All data collected by robots has to be sent to the lander, that is placed in a specific waypoint. Taking in account the large number of possibilities, we have selected a subset of significant values for the different parameters (horizon and predicate) and of experiments variables (probability of any action failure) to test our technique:

- We have considered four predicates for removal: `have_image`, `have_soil_analysis`, `have_rock_analysis` and `at`. In this paper, we only show results for predicates `have_rock_analysis` and `at`, because the results generated with the predicates `have_image`, `have_soil_analysis` are similar to those obtained by removing the predicate `have_rock_analysis`.
- We have selected four different horizons for each predicate. $k = 3, 5, 10$ and 20.
- The probability of an action execution failure is included in the definition of the action in then PPDDL domain. We show results with 60% and 30% of failure probability. Note that Abstract-MFF does not have access to the probabilistic version of the domain.

In the Rovers domain, Abstract-MFF greatly reduces the computational cost over Metric-FF when predicate `have_rock_analysis` is removed. Table 1 and 2 compares the planning time (average of all runs of the sum of all planning times during the whole re-planing loop of each run) when using abstraction with different values of k and a different probability of failure.

On one hand, we have observed that removing a predicate that is part of the preconditions of the actions that achieve the problem goals provides better performance in difficult problems, regardless of the level of stochasticity of the environment. This reduces the difference on time between the original planner and Abstract-MFF. Abstracting this kind of predicates allows the planner to achieve goals more easily, reducing the number of actions required to find a state where goals are true. For instance, if we remove the predicate `have_rock_analysis` in the action comunicate_rock_data presented in Figure 1, the number of actions necessary to get a rock sample and send it to the lander will decrease, because the robot does not have to move to the waypoint where the rock is and it does not have to pick the rock sample up to analyze it; the rover only needs to send information to the lander.

```
(:action comunicate_rock_data
   :parameters (?r - rover ?l - lander ?p - waipoint
?y - waypoint)
   :precondition
   (and
      (at ?r ?x) (at_lander ?l ?y)
      (have_rock_analysis ?r ?p) (visible ?x ?y)
      (available ?r) (channel_free ?l)
   )
   :effects
   (and
      (not (available ?r)) (not (channel_free ?l))
      (chanel_free ?l) (communicate_image_data ?o ?m)
      (available ?r)))
```

Figure 1: Action comunicate_rock_data in PDDL.

On the other hand, in Tables 3 and 4 compare planning time when predicate `at` is removed. In this case, results are worse than those for the other predicate. Our initial assumption was that predicates like `at` in domains where a robot or a vehicle has to move through different positions would offer better performance than the rest of predicates. But, in the Rovers domain this is not the case. Thus, the decision on which predicates to remove is important to obtain a good performance.

Regarding the value of k, the results shown in the different tables for the Rovers domain indicate that there is no general rule about the influence of the value of k in the performance of the planner. In some problems, lower values of k are better and in others it is the contrary. We hypothesize that the efficiency also depends on the percentage of action failures or the problem to solve. Initially, we expected that small values of k tend to increase the number of re-planning steps, since a bigger part of the plan is abstracted, but planning time will be less, given that the plan is shorter on each planning episode. However if the percentage of actions failures is high, the k value do not offer any advantage over the number of planning episodes. Because if a failure appear on the first actions of the plan, the k value only decrease the planning time, but it generates a similar number of planning episodes, without been affected by the value of k.

In general, results obtained in the Rovers domain confirm our initial hypothesis: in problems where planning time is high, abstractions significantly reduce planning time. The first group of problems are moderately difficult and abstraction does not show a significant reduction of planning time. But, in hard problems as 25 or 27, Abstract-MFF obtains a reduction of the total planning time of 60% when removing predicate `have_rock_analysis`. In addition, when two predicates (`have_rock_analysis, have_image`) are used to generate abstractions in the Rovers domain 5, the planning time is reduced in an order of magnitude. We would like to explore in the future other combinations of predicates to remove.

Results in Other Domains

We report the total planning time, as well as the mean of replanning steps. We have selected some benchmark domains: Depots, DriverLog and Satellite domains from IPC-3 and Gold-Miner domain from the learning track of the IPC-6. For each domain, we have selected two problems, two predicates to remove and four different horizons ($k = 3, 5, 10$ and 20). The probability of an action execution failure has been set to 30%.

Table 6 shows the results. Our approach obtains good results in easy and difficult problems. On one hand, it has good performance in easy problems (13 and 16) in the Depots domain; it needs less planning time if it abstracts predicates `in` and `available`. A similar behaviour can be observed in the Satellite domain, whose results present a similar performance as in the Rovers domain, where it obtained good results in hard problems.

On the other hand, in the Gold Miner domain, abstractions can solve problems avoiding dead-ends. When predicate `robot-at` is removed, the search process can avoid dead-ends when the abstraction is applied before the dead-end appears. Removing this predicate allows the robot to execute any action at every location, decreasing the complexity of the problem. However, if predicate `holds-bomb` is removed, problems can only be solved when the value of k is high, because the predicate `holds-bomb` indicates when a robot is holding a bomb. If the number of actions to hold a bomb is greater than k, the robot can enter in a cycle using the abstraction to reach the goals in the last part of the plan.

Finally, these abstractions do not offer good results in all problems. If we remove the predicate `in` in the DriverLog domain, the cost of solving the problem increases, because predicate `in` defines the position of the trucks and how they move among different locations. When it is deleted, it is possible to move to any location from every other location. Thus, we hugely increase the branching factor. We observed a similar behavior in the Gold Miner Domain, when predicate `robot-at` is removed.

Related Work

This work focuses on applying abstractions over Automated Planning to reduce the computational overhead in stochastic or dynamic environments. There have been already many approaches that generate abstractions. We review some works based on abstraction in classical automated planning and motion planning, which is closely related to our work. The first work in abstractions (Sacerdoti 1972) extended the work of Newell and Simon on GPS and used it to develop Abstrips, which combined abstraction with STRIPS. They defined abstraction levels by assigning criticalities to predicates, which define the difficulty of achieving them. The planner used these criticalities to iteratively generate successive abstract plans using only predicates in the corresponding abstract space. Alpine (Knoblock 1991) automatically generates abstraction hierarchies, using the preconditions of operators. In both cases abstractions are used to generate an abstract plan. This is refined incrementally until the lowest level of criticality is expanded and goals have been satisfied. Instead, our approach generates an abstract plan the first k where actions are valid ones, while the rest of the plan is not necessarily valid.

In recent years, abstractions have been used to generate heuristic techniques. Hoffmann and Nebel (Hoffmann 2003) used abstractions to compute the heuristic value of nodes by building a relaxed plan to guide the search process, where the delete effects of actions are ignored. Another use of abstractions to build heuristics are pattern databases (PDBs), which have been shown to be very useful in several hard search problems (Culberson and Schaeffer 1998) and Automated Planning (Edelkamp 2001). VRP has been designed to generate abstract plans in stochastic environments by decreasing computation time regardless of the technique used to guide the search process.

More recently, changes in the representation have been used to automatically generate finite-state controllers from models (Bonet, Palacios, and Geffner 2009). This represents a kind of contingent problems where actions are deterministic and some fluents are observable. The controllers could be considered general solvers in the sense that they do not solve

Prob	Metric-FF			AMFF (k=3)			AMFF (k=5)			AMFF (k=10)			AMFF (K=20)		
	Time	SD	FT	Time	SD	FT	Time	SD	FT	Time	SD	FT	Time	SD	FT
20	91	6	2	68	10	1	80	10	2	**53**	4	1	57	6	1
21	299	45	12	292	54	3	269	20	3	**91**	5	3	167	7	4
22	819	35	15	**272**	14	4	308	43	3	311	15	4	349	18	4
23	2676	60	32	**687**	190	8	542	55	7	975	76	6	1149	61	7
24	408	259	17	292	58	5	307	29	5	**231**	21	7	271	20	4
25	19414	2759	263	**4392**	269	33	4712	567	74	6340	715	78	7751	398	60
26	2522	229	25	1970	327	21	2125	587	14	**648**	42	16	895	43	52
27	24062	5984	237	6981	1961	61	6975	1681	80	**4858**	1215	42	4879	777	34
28	11848	543	228	3308	313	35	3641	559	38	**3565**	465	32	4451	518	35
Total	62143			18257			18967			**17086**			20293		

Table 1: Planning time for the Rovers domain when removing predicate `have_rock_analysis` with a 60% probability of failure. The first column corresponds to one IPC problem of the Rovers domain, the second column corresponds to Metric-FF and the rest corresponds to Abstract-MFF with different values of k. For each planner, each column shows the average of the sum of planning times of each run in seconds, the standard deviation (SD) in seconds and the time of the first planning process (FT) in seconds. In bold, we highlight the best results per row.

Prob	Metric-FF			AMFF (k=3)			AMFF (k=5)			AMFF (k=10)			AMFF (K=20)		
	Time	SD	FT	Time	SD	FT	Time	SD	FT	Time	SD	FT	Time	SD	FT
20	32	4	2	**20**	3	1	26	6	2	21	10	1	27	9	1
21	**52**	16	12	97	9	3	64	20	3	52	40	3	69	22	4
22	192	26	15	100	11	4	**83**	14	3	99	46	4	108	39	4
23	528	193	32	260	29	8	**153**	21	7	177	113	6	170	71	8
24	136	10	17	104	10	5	90	12	6	**85**	49	7	104	37	5
25	9657	932	263	1880	464	35	**1564**	219	72	1567	813	78	1796	621	58
26	680	42	25	637	42	23	**599**	160	15	662	329	16	617	292	54
27	4581	674	237	2187	438	62	1874	585	78	**1554**	887	44	1989	621	35
28	3132	223	168	1076	250	33	949	175	39	**929**	461	31	1181	362	36
Total	18983			6365			5385			**5149**			6065		

Table 2: Planning time for the Rovers domain when removing predicate `have_rock_analysis` with a 30% probability of generating an unexpected state. The meaning of columns is the same as previous table.

only the original problem, but also can include changes regarding to the size of the problem or the probabilities of the action effects.

The concept of abstraction has been explored across many other areas of AI. Our work is inspired by the work of Zickler and Veloso (Zickler and Veloso 2010) in motion planning. This technique generates a collision-free trajectory from an initial state to a goal state in dynamic environments. This work considers the far future with a different level of detail, selectively ignoring the physical interactions with dynamic objects of environment. We try to apply a similar idea in automated planning and analyze effects of ignoring any far future information about the environment and decrease complexity of the problem to get a better performance.

Conclusions

In this paper, we have presented Abstract-MFF, a planner based in Metric-FF that uses an abstraction mechanism that dynamically removes some predicates during the planning process based on a temporal horizon, in order to improve planning performance in stochastic environments. The main contribution of this work consists on understanding the effects of this idea over the planning process, and how it can be used to improve the application of planning techniques in real environments.

Regarding the experimentation, we observe that the planning time is most of the time less than the time required by Metric-FF. Also, the time needed to generate the first plan is always lower, which in real environments is critical, as it allows the execution to begin earlier. Besides, the cost of the execution (number of executed actions) using the presented technique is roughly the same as the one of Metric-FF, with an overall improvement of a 5%. Furthermore, the experiments hint that combining different abstracted predicates may lead to an increase in efficiency. As further work, we would like to explore which combinations of predicates are useful to be removed, automatic ways of selecting those, and changing the temporal horizon dynamically. Also, it would be interesting to select a set of problems that cannot be solved by current planners like Metric-FF (Hoffmann 2003) or LAMA (Richter and Westphal 2008) and analyze if our technique can solve these problems applying abstrac-

Prob	Metric-FF			AMFF (k=3)			AMFF (k=5)			AMFF (k=10)			AMFF (K=20)		
	Time	SD	FT	Time	SD	FT	Time	SD	FT	Time	SD	FT	Time	SD	FT
20	91	6	6	42	10	1	**39**	5	1	53	1	1	573	1	2
21	299	45	12	314	54	3	282	39	3	**121**	7	3	177	15	2
22	819	35	15	445	14	4	450	43	4	**333**	29	4	355	28	4
23	2676	60	32	1428	190	8	1530	55	4	**965**	30	6	1509	88	7
24	408	259	7	308	58	5	327	29	5	**254**	31	6	255	33	6
25	19414	2759	263	18108	269	31	12652	567	80	**6370**	285	73	7791	151	60
26	2522	129	25	1493	327	19	1418	587	18	**668**	53	12	2563	3658	52
27	24062	5984	237	7427	1961	62	10815	1681	81	**4718**	449	41	4799	382	34
28	11848	543	228	10973	313	35	9491	274	37	**3505**	313	32	4391	335	42
Total	62143			40544			37004			**16987**			22413		

Table 3: Planning time for the Rovers domain when removing predicate `at` with a 60% probability of failure. The meaning of the columns is the same as previous table.

Prob	Metric-FF			AMFF (k=3)			AMFF (k=5)			AMFF (k=10)			AMFF (K=20)		
	Time	SD	FT	Time	SD	FT	Time	SD	FT	Time	SD	FT	Time	SD	FT
20	32	4	2	**20**	3	1	26	6	2	21	10	1	27	9	1
21	**52**	16	12	97	9	3	64	20	3	52	40	3	69	22	4
22	192	26	15	100	11	4	**83**	14	3	99	46	4	108	39	4
23	528	193	32	312	34	18	321	84	19	**297**	42	14	301	59	15
24	136	10	17	124	11	12	129	72	13	**103**	29	10	119	46	12
25	9657	932	263	6721	424	113	6544	328	104	**5567**	743	98	5996	832	97
26	680	42	25	797	87	30	691	315	28	671	279	25	**624**	79	37
27	4581	674	237	3971	428	188	3874	498	166	**2954**	731	110	2973	621	137
28	**3132**	223	168	4184	328	179	3949	315	142	4128	537	138	3997	512	168
Total	18983			16290			15681			**13892**			14214		

Table 4: Planning time for the Rovers domain when removing predicate `at` with a 30% probability of failure. The meaning of columns is the same as previous table.

tions. We would also like to apply this technique to robotics. In this case, we should define the interactions between the deliberative and reactive levels taking into account the capabilities and constraints of the robots and the environment, as in (Lematre and Verfaillie 2007). In our case, it will be necessary to analyze how abstractions are done when dealing with different level of reasoning.

Acknowledgments

We extend our thanks to Vidal Alcázar and Raquel Fuentetaja for their helpful discussions. This research is partially supported by the Spanish MICINN projects TIN2008-06701-C03-03, TIN2011-27652-C03-02, TRA-2009-008 and Comunidad de Madrid - UC3M(CCG10-UC3M/TIC-5597). The first author is supported by a PhD grant from Universidad Carlos III de Madrid.

References

Bonet, B., and Geffner, H. 2004. A probabilistic planner based on heuristic search. In *In Proceedings of the Fourth International Planning Competition*.

Bonet, B.; Palacios, H.; and Geffner, H. 2009. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *19th International Conference on Automated Planning and Scheduling*. Thessaloniki, Greece: AAAI Press.

Borrajo, D., and Veloso, M. 2012. Probabilistically reusing plans in deterministic planning. In *Proceedings of ICAPS'12 workshop on Heuristics and Search for Domain-Independent Planning*. Atibaia (Brazil): AAAI Press.

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):318–334.

Edelkamp, S. 2001. Planning with pattern databases. In *In Proceeding of the sixth European Conference on Planning*, 13–24.

Fox, M.; Gerevini, A.; Long, D.; and Serina, I. 2006. Plan stability: Replanning versus plan repair. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling*, 212–221.

Hansen, E. A., and Zilberstein, S. 2001. Lao* : A heuristic search algorithm that nds solutions with loops. *Articial Intelligence* 129(1-2):35–62.

Hoffmann, J. 2003. The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research* 20:291–341.

Knoblock, C. 1991. Characterizing abstraction hierarchies

Prob	Metric-FF			AMFF (k=3)			AMFF (k=5)			AMFF (k=10)			AMFF (K=20)		
	Time	SD	FT	Time	SD	FT	Time	SD	FT	Time	SD	FT	Time	SD	FT
23	2676	193	32	**532**	74	3	560	76	3	474	56	4	787	51	4
25	19414	932	163	**2244**	321	16	2468	342	31	3173	293	29	4531	534	29
27	24062	674	167	**2735**	294	11	2893	301	11	3171	264	11	5028	253	13
28	11848	223	127	**1102**	104	8	1455	123	11	1885	103	13	2343	157	14
Total	18983			**6613**			6966			8703			12689		

Table 5: Planning time for the Rovers domain when removing predicate `have_image` and `have_rock_analysis` with a 60% probability of failure. The meaning of columns is the same as previous table.

Domain	Problem	Predicate	Metric-FF	AMFF (k=3)	AMFF (k=5)	AMFF (k=10)	AMFF (K=20)
			Time	Time	Time	Time	Time
Depots	13	In	**0.90**	1.20	1.85	2.15	1.60
Depots	13	Available	**0.90**	1.31	1.78	2.21	1.92
Depots	16	In	220.10	45.30	**42.10**	108.21	131.43
Depots	16	Available	220.10	**85.25**	86.25	134.92	149.12
Satellite	27	Calibrate	12854.32	4828.85	4320.78	3745.83	**3397.15**
Satellite	27	Power On	12854.32	3951.60	2769.55	**3289.32**	3653.10
Satellite	29	Calibrate	16542.91	**4467.32**	4582.43	4832.12	5104.34
Satellite	29	Power On	16542.91	3934.42	**3582.43**	4232.12	4604.34
Driver Log	14	Empty	10.98	**4.67**	5.32	6.32	5.12
Driver Log	14	In	10.98	8.73	9.21	**8.19**	9.42
Driver Log	19	Empty	1828.52	**460.80**	878.34	793.29	620.31
Driver Log	19	In	**1828.52**	8720.43	8892.82	9621.64	9053.53
Goldminer	12	robot-at	-	308.82	235.74	**82.10**	-
Goldminer	12	holds-bomb	-	-	-	-	**45.32**
Goldminer	15	robot-at	-	395.45	318.09	**75.53**	-
Goldminer	15	holds-bomb	-	-	-	-	**89.68**

Table 6: Planning time for different domains and problems. The columns correspond to the domain, problem number, the removed predicate, the results of Metric-FF and the results of Abstract Metric-FF with different values of k. Each row corresponds to a problem of a domain. For each planner, we provide the average time of execution in seconds. In bold, we highlight the best results.

for planning. In *In Proceedings of the Ninth National Conference of Artifical Intelligence*.

Lematre, M., and Verfaillie, G. 2007. Interaction between reactive and deliberative tasks for on-line decision-making. In *17th International Conference on Automated Planning and Scheduling*. Providence, Rhode Island, USA: AAAI Press.

Peot, M. A., and Smith, D. E. 1992. Conditional nonlinear planning. In Kaufmann, M., ed., *In Proceedings of the First International Conference on Artificial Intelligence*, 189197.

Richter, S., and Westphal, M. 2008. The lama planner using landmark counting in heuristic search. In *Proceedings of the International Planning Competition*.

Sacerdoti, E. D. 1972. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence* 2(5):115–135.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. Ff-replan: A baseline for probabilistic planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*.

Younes, H. L. S., and Littman, M. L. 2004. Ppddl1.0: An extension to pddl for expressing planning domains with probabilistic effects. *In Technical Report CMU-CS-04-162*.

Younes, H. L. S.; Littman, M. L.; Weissman, D.; and Asmuth, J. 2005. The first probabilistic track of the international planning competition. *Journal of Artificial Intelligente Research* 24:851–887.

Zettlemoyer, L. S.; Pasula, H.; and Kaelbling, L. P. 2005. Learning planning rules in noisy stochastic worlds. In *In Proceedings of the Twentieth National Conference on Artificial Intelligence*, 911–918.

Zickler, S., and Veloso, M. 2010. Variable level-of-detail motion planning in environments with poorly predictable bodies. In *In Proceeding of the nineteenth European Conference on Artificial Intelligence*.

Branched Plans with Execution-Time Choice to Reduce the Costs of Conservative Planning

Amanda Coles and **Andrew Coles**
Department of Informatics,
King's College London,
London, WC2R 2LS, UK
email: {amanda,andrew}.coles@kcl.ac.uk

Abstract

In many applications, especially autonomous exploration, there is a trade-off inherent in the way resource usage is estimated at planning-time: to maintain the safety of the agent, the plan must be robust to pessimistic outcomes; yet, to maximise the amount of activities undertaken, resources must be used to their full potential. In this paper we consider a method of planning that maintains operational safety, by detailing additional plan fragments to be performed at execution-time on an 'opportunistic' basis, if conditions permit. We consider planning problems with soft goals, each with an attached cost paid if the goal is not achieved. Our plan representation is a tree structure, with simple execution-time rules: take the best-cost branch, given some associated resource level constraint. We demonstrate the use of such plans can potentially dramatically increase utility, whilst still obeying strict safety constraints.

1 Introduction

Opportunities for communication with remote autonomous agents are often scarce, whether in space, underwater, or in disaster-recovery environments. The ideal of on-board planning is currently difficult to achieve due to two primary factors: the reluctance of controllers to trust fully autonomous behaviour and the processing and memory constraints of remote agents. It is therefore necessary to provide agents with plans for long periods, whenever communication is possible.

In such situations conservatism is ubiquitous: the desire for continued safe operation of autonomous exploration agents restricts the amount of exploration that can be performed. To give an example, it is estimated that the Mars exploration rover Sojourner spent 50% of its time on the surface idle as a result of either having completed all planned activities, or due to plan failure (Fox et al. 2006). Space agencies often generate plans on the ground, primarily by hand with supporting software, using highly conservative estimates of energy consumption (Meuleau et al. 2009).

In this work, we consider the problem of creating plans that are cost-effective, whilst adhering to the strict safety constraints required. We consider *over-subscription* problems, where each goal has an associated cost, incurred if it is not reached. Such goals may arise, for instance, from the many competing science activities a Martian rover could perform. We first extend a forward-chaining planning approach to support uncertainty in the numeric effects of actions, and confidence thresholds on numeric conditions. The resulting planner is capable of optimising quality in terms of the goal costs, whilst ensuring the plan completes with the requisite degree of confidence.

Using such a planner it is possible to set a high confidence threshold, and find a solution that will succeed under a wide range of execution outcomes. This is both a strength, and a weakness: the plan is statistically likely to succeed, but is also likely pessimistic. At execution time, we have additional knowledge — we know how much resource past actions consumed — and although we must remain pessimistic about the future, we may reach a point where a lower-cost goal state could be reached, with acceptable confidence. If the plan is fixed, though, execution will not exploit this.

As on-board replanning is often not possible, it is useful to attempt to predict where potential resource excesses could lead to lower costs, and generate *conditioned branches* for use at execution time, if conditions allow. As high-confidence plans are pessimistic, it is likely such surpluses exist, and hence, at execution time, such branches will often be used. Our approach gives the advantage of maintaining control over operations (only a finite space of plans could be executed), whilst allowing better costs through exploiting opportunities that arise during execution. This is related to the idea of creating policies, but differs in that we do not have to generate complete policies for all eventualities. To evaluate our approach, we compare to a single pessimistic plan, a simulation of what could be achievable by on-board replanning, and make an indicative comparison to a policy-based approach. Our results show improved utilities with respect to a single-plan approach, and indicate scalability with respect to policy based approaches.

2 Background

Here we define formally the problem we are solving and compare existing approaches in the literature to solving related problems.

2.1 Problem Definition

A planning problem Π is a tuple $\langle F, \mathbf{v}, I, G, A, C, \theta \rangle$ where:

- F is a set of propositional facts;
- $\mathbf{v}$ is a vector of numeric variables;

- I is the initial state: a subset of F and assignments to (some) variables in $\mathbf{v}$;
- A *condition* is a first-order logic formula over facts in F, and *constraints* on the values of $\mathbf{v}$. Such constraints are Linear Normal Form (LNF) formulæ:

$$(\mathbf{w}.\mathbf{v}\ op\ l)$$

  ...where $op \in \{>, \geq\}$; $l \in \Re$; and $\mathbf{w}$ is a vector of real values.
- G describes the goals: a set of conditions. Each $g \in G$ has an associated cost $c(g) \in \Re^+$ if g is not true at the end of the plan.
- A is a set of actions each with:
  - $Pre(a)$: a (pre)condition on its execution;
  - $\mathit{Eff}^-(a)$, $\mathit{Eff}^+(a)$: propositions deleted (added) upon applying a;
  - $\mathit{Eff}^{num^d}(a)$: a set of numeric variable updates that occur upon applying a. Each is of the form $\langle v\ op\ D(\mathbf{v}, params)\rangle$ where $op \in \{\text{+=}, \text{=}\}$ and D is a parameterised probability distribution that governs the range of outcomes of the effect, given $\mathbf{v}$. (For deterministic effects, D is deterministic.)
- C is a set of global conditions: each $c \in C$ is a condition.
- θ is the required confidence level on successful plan execution: the constraints C are always true, and when an action is to be executed, its preconditions are met.

We adopt the state progression semantics of the planner RTU (Beaudry, Kabanza, and Michaud 2010). A Bayesian Network is used to define the belief of each $\mathbf{v}$, and as actions are applied, the network is updated with additional variables. In a state S_i, for each $v_j \in \mathbf{v}$, a variable v_i^j is associated with the belief of v. If an action a is applied, leading to a state S_{i+1}, then for each effect numeric effect $\langle v^j\ \ op\ \ D(\mathbf{v}, params)\rangle$, two random variables are added to the network. The first of these, D_{i+1}^j, represents $D(\mathbf{v}, params)$. The second, v_{i+1}^j, is associated with the belief of v in S_{i+1}, and it is determined by either:

- $v_{i+1}^j = v_i^j + D_{i+1}^j$, if op is +=;
- $v_{i+1}^j = D_{i+1}^j$, if op is =.

For each variable unaffected by a, the network variable associated with the belief of the variable is unchanged.

The Bayesian network is key to determining whether a plan meets the required confidence level θ. An action a is applicable in a state S_i if $Pre(a)$ is satisfied; and a given state is valid if all the conditions C are met. A sequential solution to Π is then a sequence of steps $[a_0, .., a_n]$, implying a state trajectory $[I, S_0, .., S_n]$. We require that with $P \geq \theta$, in a given execution of the plan, all states are valid, and each action's preconditions are met. In the general case, this may require Monte-Carlo sampling of the network, but for some subsets of the input language, this may be performed analytically. The cost of this solution is the sum of $c(g)$ for goals not true in the terminal state.

2.2 Related Work

One popular approach to solving planning problems with uncertainty, including those with resource-usage uncertainty, is to use Markov Decision Processes (MDPs). Meuleau et. al. (2009) considered the problem of maximising reward in over-subscription planning using an MDP approach and building a complete policy. MDPs are popular as they can offer optimality guarantees that other approaches can not. However, this comes at a price: increased computational effort compared to more classical approaches; and despite recent improvements (Mausam and Weld 2008; Rachelson et al. 2008) scalability is increasingly challenging when continuous resources and time are involved. The size of the policies produced is also related to our work: an MDP, in encoding many possible plan trajectories, is less scrutable than if fewer options are maintained, a limitation when operations staff wish to maintain tight control and confidence in the system, to be certain about the agent's safety.

A closely related (non-MDP) approach is that of the planner RTU (Beaudry, Kabanza, and Michaud 2010) which uses forward-chaining search to generate plans to achieve a (fixed) set of goals. The plans found complete to within a deadline to a certain confidence level, given the distributions on resource/time usage, and optimising some combination of makespan and cost. We build on these techniques, addressing the additional challenges of maximising utility in the presence of soft goals, and considering how a plan with branches can allow execution-time conditions to dictate which actions to execute. Also related is the planner Tempastic (Younes and Simmons 2004): as in our work, it begins with a single plan, and then augments it given the uncertainty in the problem. The problems being considered though are very different: Tempastic considers exogenous events and actions with probabilistic propositional effects, but not continuous numeric variables and uncertain numeric effects. Related work in scheduling has considered building branches 'just in case' (Drummond, Bresina, and Swanson 1994): this takes the opposite view to our work generating optimistic schedules and building alternatives for the case where the schedule would otherwise fail.

Prior work has considered the development of tools to assist human activity planners in generating plans on the ground (Fox et al. 2006). The plan validator, VAL (Howey, Long, and Fox 2004), was used to note errors in plans; to suggest plan repair options for use if part of the plan fails to execute; and to suggest diagnostic actions for inclusion in the plan. In this work all repair suggestions were to be implemented by the human planners. Despite the limited nature of the permitted suggestions, the approach was shown to have greater potential than the established approach of receiving a failure report one sol, uploading a diagnostic plan the next, and finally resuming operations on the third.

When considering the execution of plans, various executives support plan structures that contain an element of choice. A relevant recent piece of work is the Drake system (Conrad and Williams 2011). Here, plans are represented as Disjunctive Temporal Networks, labelling constraints with the consequences of the (limited set of) choices that can be made. In effect, as in our approach, it compactly

represents a small number of plans with differing properties and requirements, such that the executive then has flexibility as to which choices to make (i.e. which actions to dispatch) given runtime conditions.

Several approaches have been proposed for deterministic over-subscription planning problems. These include work on problems where the cost constraints are solely in this form (Smith 2004; Edelkamp and Kissmann 2008, Benton et al. 2009) and more general approaches for PDDL3 (Gerevini et al. 2009) preferences (Edelkamp et al. 2006; Baier et al. 2007; Coles and Coles 2011).

3 Over-Subscription Planning under Resource Uncertainty

Over-subscription planning problems are characterised by a surfeit of goals, and some sort of means for determining which combinations of goals are preferable to others. Each goal g is assigned a cost $c(g)$, and the metric cost of a plan is the sum of the costs of the goals which it does not reach. One plan is then preferable to another if its metric cost is lower. When planning with resource uncertainty, we have the additional consideration to make that some plans are more or less likely to complete. There is an inherent trade-off: a good high-confidence plan will be more conservative and hence have worse plan cost than a good less-confident plan.

In this section, we explore the issues arising where over-subscription and uncertainty meet, and the confidence–cost trade-off. First, we detail how we adapt a forward-chaining search approach for over-subscription planning, to consider the uncertainty in effects on numeric variables and to ensure the plan succeeeds with the desired confidence. Second, we discuss a compromise between a single, linear solution, and a full-policy solution to this class of problems, extending an conservative initial plan with branches for use at execution time if conditions are suitable.

3.1 Adapting Forward-Chaining Search

In order to effectively use a forward-chaining approach for the class of problems being considered in this work, two important considerations to make are how to manage uncertainty during search, and which heuristic to use.

For the first, we turn to the planner RTU (Beaudry, Kabanza, and Michaud 2010) and its Bayesian Network approach, described earlier in Section 2.1. For a given plan, the Bayesian network captures the distribution of variables' values in each of the states along the plan trajectory, given the effects of the actions. Then, at each state during search, we can query the network to ensure the plan will succeed acceptably often: as noted in Section 2.1, with $P \geq \theta$, each state S must satisfy the conditions C, and if an action a is applied in S, S must further satisfy any preconditions of a.

This part of the approach does not change fundamentally with the shift to over-subscription planing. Rather, what is more involved is the heuristic guidance needed. As in the case where all goals are hard, we need some sort of estimate of 'actions to go' until all goals are met. Further, as some goals might not be reachable from a given state, we would like to identify this too: if we have already have an incumbent solution with some cost, but carry on searching, we can prune states based on knowledge of unreachable soft-goals, i.e. reachable cost. To serve both of these purposes, we take as our basis the non-LP heuristic proposed used in LPRPGP (Coles and Coles 2011): a variant of the Metric Relaxed Planning Graph (RPG) heuristic (Hoffmann 2003), extended to handle PDDL3 preferences. As the 'soft goals' in this work are a subset of PDDL3 (corresponding to goal preferences) it suffices to describe the heuristic as follows:

1. The RPG begins with fact layer zero, $fl(0)$: the facts and variable values in the state S being evaluated.
2. Action layer i, $al(i)$, contains actions applicable in $fl(i)$;
3. $fl(i{+}1)$ is derived from $fl(i)$ by *relaxing* the effects of the actions in $al(i)$: delete effects are ignored, and optimistic upper/lower bounds on numeric variables are kept.
4. The RPG is expanded by adding alternate fact and action layers following these rules.
5. Graph expansion terminates at the first fact layer where expanding the planning graph further would not lead to more goals or soft-goals being achieved.
6. A relaxed plan is extracted, containing actions to meet each of the goals that appeared in the RPG.

It is important to note that at point 5 here, graph expansion only stops when each goal has been met, or has been proven to be unreachable even under the relaxed semantics. Thus, if a goal does not appear, it cannot be satisfied in any state reached from S. This is a rich source of heuristic knowledge about the cost of reachable states: if the metric comprises only a weighted sum of binary variables denoting whether each goal is achieved, an admissible estimate of the cost of reachable states is the sum of cost of the goals not reached during RPG expansion. Then, as discussed above, if search is bounded by the cost of an incumbent solution, any state with an admissible cost in excess of the cost of this can be pruned: it cannot possibly lead to a better solution.

The original heuristic described above does not directly refer to uncertainty: it assumes variables have known values, and effects have known outcomes. As such, we must modify it to be suitable for our purposes. First, we must define the values of the variables in fact layer zero. For this, we turn to the Bayesian network: the value of each $v \in \mathbf{v}$ is taken by querying the network to find the mean value of v in S. This is a single, fixed value, suitable for the RPG as described. Second, for each numeric effect, we assume it has its mean outcome. Third, if a precondition can be reached in the RPG, we assume it can be satisfied 100% of the time. If $\theta \geq 0.5$, then from Jensen's inequality we known that, in effect, we have 'relaxed' the uncertainty: the heuristic is optimistic, so somehow restoring uncertainty would not allow more goals to be met. With reference to state pruning, this is an important property to maintain: it is not reasonable to prune a state on the basis of what was unreachable in the heuristic if, actually, it may in fact be reachable.

3.2 Opportunistic Branching

This forward-chaining search approach finds a sequential solution plan to a planning problem which, statistically, will

respect each constraint, given the uncertain nature of execution. When planning with a high degrees of confidence, for instance, θ=0.999, the resulting plan is necessarily quite conservative. It will still occasionally fail (with $P < 0.001$) but on average, the plan will not come close to violating its constraints and may therefore compromise cost.

An alternative to finding a linear solution plan, addressing this limitation, is to find a policy: state–action pairs that, beginning with the initial state, dictate which action to execute. In the presence of continuous variables, some sort of approximation is necessary, with each policy state representing a number of reachable states. Otherwise, in theory, when applying an effect whose outcome is governed by some distribution, an infinitely large number of states is reached, identical modulo different values of the variable altered. A linear plan is a coarse approximation, where all the states reachable after an action are collapsed into single policy state, associated with which is the next step of the plan. Such a representation is compact, but as discussed, has its limitations. More sophisticated approaches such as (Mausam and Weld 2008) use discretization approaches, where applying an action in a state will lead to one of a finite number of policy states. Such policies have better cost performance than a linear plan, but are considerable in size, with scalability being the main limitation of such approaches.

As a compromise measure between these, we build a partial policy. The spine of the policy is a linear plan that, with $P \geq \theta$, will execute successfully. Attached to this are branches for *opportunities* which, if execution-time condition permits, can be followed to reach a lower-cost goal state. The structure of such plans can be represented naturally as tree $\langle V, E\rangle$. Each $v \in V$ is an action from A, with v^0 (the root node of the tree) corresponding to the first step in the plan. Each $(i, j) \in E$ is labelled with one or more condition–cost pairs, $\langle f_k, c_k\rangle$, where:

- After applying the action v^i, if the state S_i reached satisfies one of these conditions f_k, execution may continue with step v^j;
- If there are several $(i, j) \in E$ with at least one condition satisfied, some criterion is used to select a single v^j. We select (arbitrarily) one of:

$$\arg\min_{(i,j)\in E} \min\{c_k \mid \langle f_k, c_k\rangle \in \mathit{labels}(i, j) \ \wedge S_i \vDash f_k\}$$

Each f_k is derived by computing, using the Bayesian network, the weakest preconditions of a plan with cost c_k rooted at v^j. It specifies the constraints on the continuous state variables required to ensure that, statistically, if the j branch is chosen, it will execute successfully with $P \geq \theta$. As a simple example, consider a branch with a single resource-using action that has an effect of v+=$N[-10, 3]$, i.e. decreasing v by a normally distributed amount (mean 10, standard deviation 3). If θ=0.99 and there is a condition $c \in C$ that states $(v \geq 0)$, this must be true with $P \geq 0.99$ after the effect has occurred. Thus, the weakest precondition of this branch is calculated as the smallest value of v for which this holds: approximately, $v \geq 19.35$.

Algorithm 1 outlines our branch-planning approach. Initially, we call $\mathit{BranchPlan}(I, \infty)$, and hence at line 1, the planner is invoked. From lines 5 to 15, the steps of the plan found are considered in turn. Line 7 is key to our branch-generation approach: each variable is set to its 50th percentile, i.e. assuming resource change so far has been nominal, rather than extreme[1]. This forms the basis of the initial state for a recursive call to $\mathit{BranchPlan}$. If this returns a non-empty plan tree, then due to the cost bound imposed, it necessarily reaches a goal state with better cost than that of Π, and the tree is merged in. As the plan tree may begin with a plan segment identical to that following i, line 12 skips over the first steps of the new plan tree whilst it remains the same as the plan that would be executed anyway. Then, any remaining portion of the tree, rooted at j, is added at the point where the plan diverges. Having built the tree, the final loop (lines 16–18) label the edges out of each step in the plan with condition–cost pairs: one such pair $\langle f, c\rangle$ for each tree traversal (i.e. plan tail) reachable from j, labelled with its weakest preconditions f, and the cost reached c.

Algorithm 1: Branch Plan

```
   Data: S, an initial state; U, a cost-bound on search
1  Π ← plan(S, U);
2  (V, E) ← tree for Π, with vertex i denoting step a_i;
3  U' ← cost of Π;
4  S' ← S;
5  for each a_i ∈ Π do
6      S' ← apply a_i to S';
7      S'' ← S' setting each variable to its mean value;
8      (V', E') ← BranchPlan(S'', U');
9      if V' is non-empty then
10         j ← root of V';
11         i' ← i + 1;
12         while i' and j have at most one outgoing edge
           ∧ are labelled with the same action do
13             increment i' and j by 1;
14         add subtree of (V', E') rooted at j to (V, E);
15         add (i' − 1, j) to E;
16 for each a_i ∈ Π do
17     for each (i, j) ∈ E do
18         label (i, j) with condition–cost pairs of plans from j;
19 return A plan tree, (V, E)
```

An example of the output from this algorithm is shown in Figure 1. The initial plan found is the right-most branch of the tree, terminating with cost 116. The two octagonal vertices denote points from which the recursive call to $\mathit{BranchPlan}$ found a solution with better cost: from the first, with cost 76.5; from the second, with cost 0. In both cases, the solutions overlapped with the existing plan, so the branches are added where they first diverge. After `com_soil_data w7 w9 w0`, we can see an example of multiple edge labels: the right-most path is labelled with two condition–cost pairs. Which path is taken depends on the value of energy at execution time: from $[219.2, 354.7)$ the left path; otherwise, the right path.

As a final remark here, we note that we assume it is reasonable to consider the plans in a sequential order. In the presence of multiple agents, this is not necessarily reason-

[1] Other percentiles from the CDFs could be used.

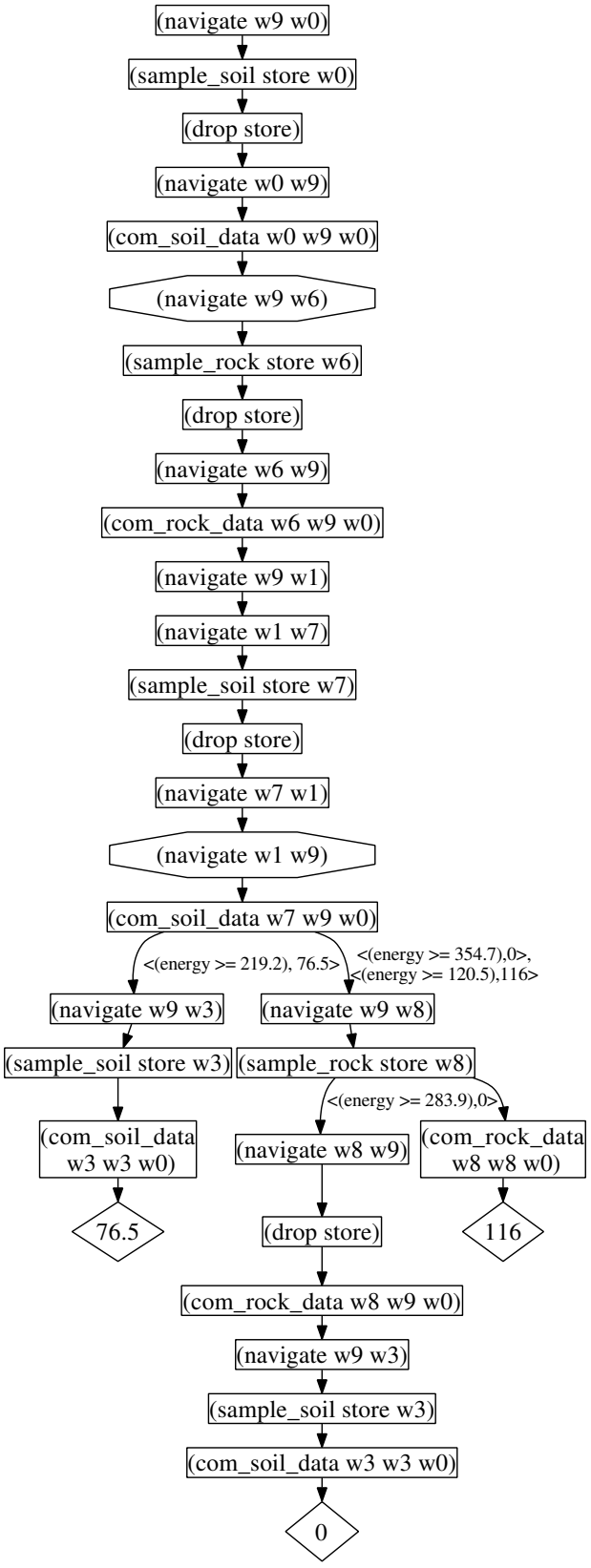

Figure 1: An Example Branched Plan

able as, conceivably, permuting non-related steps in the solution returned by the planner may lead to a more effective branch plan. For instance, if branching after a but before b leads to a cost-effective branch, but not vice-versa, this will not be found if the linear plan happens to order the steps b, a. An effective solution to this is still an open question. One option is to fix the division of goals between agents *a priori* and plan for them individually. (This would be necessary, in any case, if the agents are unable to communicate or are not orchestrated, once execution has begun.)

4 Evaluation

To evaluate our approach we investigate its performance on a Martian rover problem. We extend the over-subscription variant of the Rovers domain, introduced in the the Fifth International Planning Competition (Gerevini et al. 2009). In this domain the rover must navigate around the surface of Mars to perform several science-gathering tasks, and communicate the resulting data to a lander on the Martian surface. In the original encoding the metric, describing the costs of violating soft goals, was also augmented by the total traversal cost incurred by the plan. Since navigation can account for the majority of energy usage in real applications we used this variable to measure energy, and made each navigate action use energy according to a normal distribution with mean equal to the traverse cost and standard deviation of some constant multiplied by the traverse cost. All other actions remain unchanged. Then, we added a safety constraint to C: `(always (≥ (energy) 0))`.

When constructing linear plans in this domain, in the Bayesian networks used, the energy value in each state is equal to the initial value plus the sum of a number of normally-distributed effects. Analytically, this reduces to a single normally distributed variable (sum the means, sum the variances). As far as we are aware it is not possible to compute the expected cost of our branched plans analytically. We therefore use a Monte-Carlo approach in our evaluation to compare the initial plans and branched plans. Each plan is executed 1000 times using random sampling from the distributions to simulate the outcomes of stochastic effects.

In the branched plans, when branches are encountered, the branch to follow is selected according to the values generated in simulation and the criteria described in Section 3.2. At the end of each simulated run the plan cost is recorded: we report the average cost achieved over these 1000 runs and the number of failures. We consider a failure mode we dub 'standard' for most of this evaluation: the plan is executed as given; failure is reported if C or an action's preconditions are broken, and the cost of that run is disregarded. We consider a second mode 'twig', with execution time checks at each point, for the final hypothesis.

In order to ensure that planning terminates in a reasonable length of time we must impose time limits on both computation of the initial plan (T_i) and each branch plan (T_b). In theory an optimal plan will eventually be produced for every call of the planner; in practice, however, this only occurs in small problems as proving optimality requires exhaustion of the search space. Imposing a time limit introduces a trade-off, as restricting the time that the planner has to search for

	T_i	T_b	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
Cost(O)	900	-	370.4	294.0	370.4	58.9	326.6	198.6	126.1	278.4	192.5	169.9	**325.8**	**292.1**	602.1	180.9	1651.4	2429.8	586.1	93.1	949.1	320.7
Cost(O)	60	-	370.4	294.0	370.4	58.9	326.6	198.6	126.1	278.4	192.5	169.9	397.5	293.6	602.1	180.9	1651.4	2429.8	586.1	93.1	949.1	320.7
Cost(B)	60	10	158.3	222.0	158.3	23.5	204.5	98.5	102.2	109.3	192.5	61.6	199.9	59.7	**348.8**	180.9	766.0	1456.7	385.4	36.5	949.1	135.6
Cost(B)	60	5	158.3	222.0	158.3	23.5	204.5	98.5	102.2	109.3	192.5	61.6	199.9	59.7	365.9	180.9	766.0	1456.7	385.4	36.5	949.1	135.6
Cost(O)	10	-	370.4	294.0	512.2	58.9	501.3	198.6	278.3	278.4	192.5	169.9	397.5	293.6	602.1	180.9	1651.4	2429.8	586.1	93.1	949.1	1318.9
Cost(B)	10	10	158.3	222.0	278.7	23.5	397.4	98.5	170.8	109.3	192.5	61.6	199.9	59.7	348.8	180.9	766.0	1456.7	385.4	36.5	949.1	302.3
Cost(O)	5	-	370.4	294.0	512.2	58.9	501.3	198.6	278.3	278.4	192.5	169.9	397.5	293.6	602.1	180.9	1651.4	2429.8	586.1	93.1	949.1	1318.9
Cost(B)	5	5	158.3	222.0	158.3	23.5	204.5	98.5	102.2	109.3	192.5	61.6	199.9	59.7	365.9	180.9	766.0	1456.7	385.4	36.5	949.1	135.6
Cost(E)	900		370.4	294.0	370.4	58.9	*501.3*	*343.6*	126.1	278.4	192.5	*281.8*	325.8	284.7	585.4	180.9	3243.5	3326.4	586.1	100.5	949.1	320.7
FR(O)	900	-	1.0	0.2	1.0	0.9	0.1	0.4	0.6	0.2	0.4	1.0	0.3	0.8	0.3	0.3	0.7	0.7	0.3	0.8	0.0	1.4
FR(O)	60	-	1.0	0.2	1.0	0.9	0.1	0.4	0.6	0.2	0.4	1.0	0.2	0.7	0.3	0.3	0.7	0.7	0.3	0.8	0.0	1.4
FR(B)*	60	10	0.9	0.4	0.9	0.7	0.0	0.2	0.9	0.3	0.0	0.5	0.0	1.6	0.2	0.3	0.6	0.7	0.3	0.8	0.0	0.8
FR(B)	60	5	0.9	0.4	0.9	0.7	0.0	0.2	0.9	0.3	0.0	0.5	0.0	1.6	0.2	0.3	0.6	0.7	0.3	0.8	0.0	0.8
FR(O)	10	-	1.0	0.2	0.4	0.9	0.0	0.4	0.0	0.2	0.4	1.0	0.2	0.7	0.3	0.3	0.7	0.7	0.3	0.8	0.0	1.0
FR(B)	10	10	0.9	0.4	0.7	0.7	0.0	0.2	0.3	0.3	0.0	0.5	0.0	1.6	0.2	0.3	0.6	0.7	0.3	0.8	0.0	1.7
FR(O)	5	-	1.0	0.2	0.4	0.9	0.0	0.4	0.0	0.2	0.4	1.0	0.2	0.7	0.3	0.3	0.7	0.7	0.3	0.8	0.0	1.0
FR(B)	5	5	0.9	0.4	0.7	0.7	0.0	0.2	0.3	0.3	0.0	0.5	0.0	1.6	0.2	0.3	0.6	0.7	0.3	0.8	0.0	1.7
FR(E)	900	-	1.0	0.2	1.0	0.9	0.0	0.0	0.6	0.2	0.4	0.4	0.0	1.5	0.0	0.3	0.0	0.7	0.6	0.8	0.0	1.4
Twig Cost(O)	60	-	428.7	310.8	428.7	85.7	345.7	218.5	140.7	298.0	193.4	235.4	398.0	300.2	618.7	200.5	2033.2	2808.7	594.3	125.0	949.1	389.3
Twig Cost(B)	60	10	234.5	241.8	234.5	51.2	225.0	125.6	121.3	130.1	193.3	135.9	198.3	67.8	364.6	199.1	1277.0	1982.7	404.9	83.1	949.1	389.3
Cost(R)	60	10	189.5	189.1	190.8	55.5	213.3	120.3	99.3	124.4	118.9	87.4	195.4	49.1	316.7	188.7	1222.8	1699.8	355.3	75.4	918.5	202.2
Twig FR(O)	60	-	0.0	0.0	0.0	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Twig FR(B)	60	10	0.1	0.1	0.1	0.1	0.2	0.0	0.0	0.1	0.0	0.1	0.2	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
FR(R)	60	10	0.1	0.1	0.1	0.2	0.1	0.0	0.0	0.1	0.0	0.2	0.5	0.0	0.0	0.0	0.1	0.0	0.2	0.0	0.0	0.0
PlanNodes(O)	60	10	12	18	15	13	10	29	14	27	29	27	20	28	31	21	34	31	30	33	4	39
PlanNodes(B)	60	10	58	22	61	18	26	34	47	40	29	113	39	156	206	21	52	36	34	88	4	132

Table 1: Average Cost and Failure Rates (FR) for the Initial Linear Plan, O; the Branched Plan, B; the Equivalent-Certainty* Linear Plan (E); and the Simulation of Online Replanning (R). $\theta = 0.99; \sigma = \mu$; T_i, T_b as indicated.

plans may result in higher cost plans; though if no limit on time is upposed, planning could run (effectively) indefinitely. We explore this issue later. Tests are run on a 3GHz CPU, all calls to the planner are restricted to 3GB of RAM.

Hypothesis: the failure rate for branched plans will be higher than for the original linear plans. Table 1 shows the failure rates and average cost achieved for the original linear plans and the final branched plans. Note that although the initial linear plan generated will meet the specified confidence threshold, the branched plan may not necessarily. The table shows the failure rate for the branched plans versus the original linear plan. It is pleasing to note that although the failure rates are slightly higher, many remain within the threshold, in fact, only on problem 12 does the branched plan not meet the original threshold. In future work it may be possible to enforce slightly stricter edge constraints to ensure that the plan does meet the specified confidence level; however, for now it suffices to say that if the plan generated does not meet the desired confidence level, the initial confidence can be increased, and the planner run again, until a branched plan is generated with the desired confidence.

Hypothesis: branched plans lead to lower cost than linear plans to the same confidence (even if more time is spent generating the linear plan). Observing the anytime behaviour of LPRPG-P (Coles and Coles 2011), a planner with similar search strategy and heuristic, we can see that much of the improvement in plan quality occurs during the first 10 seconds of runtime. The results presented in that paper indicate that plans produced after 10 seconds of execution have 86% of the average quality of those found after 1800 seconds, so we therefore do not expect that limiting planning time in order to do branching will have a severe impact on the cost of the plans that can be produced. Further, we expect the branching techniques to be able to improve upon cost due to the assumed extra knowledge about the environment (estimated expected energy levels, which are checked at execution time) from which they benefit.

Comparing first the cost achieved by the initial linear plan and that achieved by the branched plan, in Table 1, we can see that on most problems the branched plan is able to achieve much lower cost plans upon execution; however, as discussed above this does also come at the cost of a slightly increased failure rate. We therefore did further experiments: setting θ to the failure rate of the branched plan, and finding a linear plan, allowing 15 minutes of planning time. The results are shown in Table 1 as Cost(E). As can be seen, a linear plan cannot deliver equivalent performance.

We also tried giving the planner θ=0.99 and 15 minutes to solve the problem, to confirm that the high cost of the linear plans found initially is not simply due to limited planning time. These are shown as cost(O) T_i = 900 in the table. There are only two problems (11 and 12) on which the cost of the original linear plan is improved by planning for 15 minutes versus 60s, and on these the cost reduction is marginal compared to the further gains made by using the branched plan. If we reduce T_i to just 10 seconds cost increases by a reasonable amount on 4 problems, suggesting that it is useful to invest more than 10 seconds in generating a good initial plan. On a similar note we investigated what happens if the time to generate each branch (T_b) is reduced from 10 seconds to 5 seconds: here there was only a marginal increase in the cost of the branched plan in a single problem given the same T_i (60). The problems solved during branching are smaller than the original problem, as some useful activity in moving towards the goals has already been performed. In this case, therefore, increasing T_b not only has a bigger impact on overall solving time, but also has less impact on cost than increasing T_i.

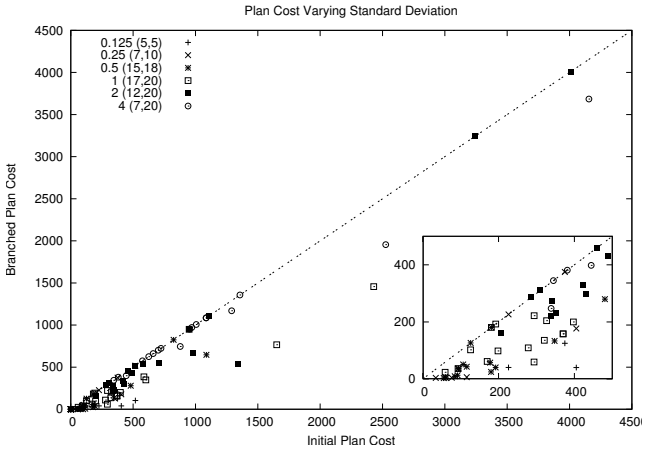

Figure 2: Effect of Standard Deviation on Cost Improvement

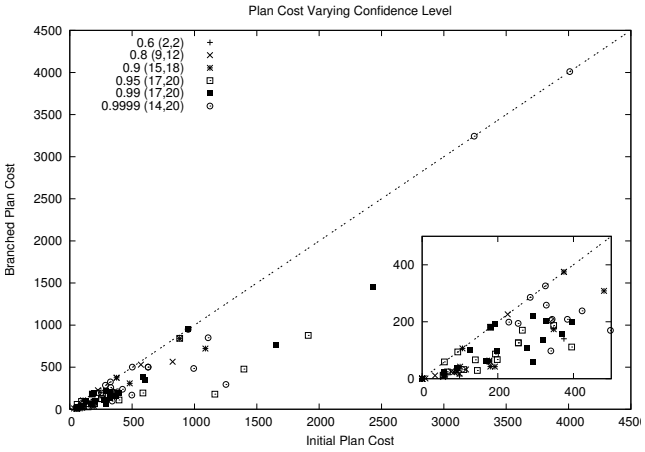

Figure 3: Effect of Confidence Level on Cost Improvement

Hypothesis: The cost decrease achieved by the branched planning process is not simply due to incidentally finding a better linear plan than the one forming the spine of the plan tree. To assess whether this is the case, for each plan tree, we enumerated all traversals from the root node, keeping only those that respect the constraints with the requisite degree of confidence, i.e. are valid linear plans. In all cases, the quality of these plans was no better than that used as the spine of the plan tree, i.e. that found by the first call to the planner. As such, we can conclude that the benefit of the branching is not simply due to finding better linear plans due to increased planning time.

Hypothesis: the standard deviation affects the cost-saving achieved by branching. Figure 4 shows the performance of the planner when varying the standard deviation (σ) of the energy consumption of navigate actions. The figures indicate the multiplier of the mean used as σ for energy consumption[2]. The numbers in brackets after each σ figure are the number of problems on which cost was improved by using branching, followed by the number of problems in which there was potential for improvement (i.e. the linear plan had non-zero cost). When σ is low there is less uncertainty, so the linear plan reaches zero more often, leaving only 5 problems in which branching can improve. The success rate for improvement is good for low σs as there needs only to be a small amount of good fortune for branches to be executable. The peak of success is in problems that are mid-range σ=μ where the problems are constrained enough that the linear plan does not solve them to zero, but there is not such a large amount of uncertainty (compared to σ=2μ) that even branching struggles to find improved plans.

Hypothesis: decreasing the confidence threshold θ will decrease the cost-saving achieved by branching. Lower confidence thresholds allow lower cost to be achieved in the initial plan. Further, the likelihood that the plan uses less resource at execution time is decreased if θ is lower. We compared several confidence levels, and the resulting improvement in plan quality between the branched and original linear plans. As can be seen in Figure 4, when θ is low the costs of the initial plans are low, so there is little scope for improvement. At θ=0.6 only two linear plans have non-zero cost, though both can be improved by branching. It is worth noting however that when θ is very high (e.g. 0.9999) the scope for improvement also decreases for a different reason: although the initial plans are not reaching cost zero, the branches are also often unable to better the cost achievable at a given state, as there is too much pessimism about future activities to make further goals achievable. Very high degrees of pessimism will by definition render very conservative overall plans that do not allow for as much extra scope.

Hypothesis: branched plans may scale better than MDP approaches It is difficult to test this hypothesis conclusively. Direct comparison to existing approaches is not possible as these either reason with uncertainty of a different nature, do not consider over-subscription planning, or are not available as runnable systems. However, we do note the results in (Meuleau et al. 2009) give indicative scalability for an MDP approach to solving similar problems. This planner was also evaluated on Rovers problems: the largest problems the planner scales to in their paper involve '1 rover, 11 locations, 20 paths and 6 goals' and take '20,000 seconds' to solve. Whilst we do not have exactly the same problems, nor the same hardware (they do not specify), an indicative comparison can be made to the size of the problems solved using our approach. The smallest problems we use are comparable to the largest solved by the MDP approach according to these parameters. The largest problems we use involve 19 locations, 60 paths and 19 goals. As a point of comparison for time-taken, when we used T_i=60s and T_b=10s, generation of the largest branched plan took just 540 seconds in total, whilst the smallest one took just over 90 seconds. This is promising for the scalability of our approach, and is to be expected since we do not have the burden of computing complete, optimal, policies.

An additional consideration is the size of the resulting plan trees. In memory-limited situations it is not possible to store large policies, therefore the size of our branched plans, which are not complete policies, could be advantageous. We cannot compare to the MDP policies above as the raw data

[2]High σs may sometimes lead to negative energy use, but we include these as the comparison remains theoretically interesting.

is not given. The 'Plan Nodes' rows of Table 1 show that the number of nodes in the branched plan for each problem remains reasonable in relation to the size of a linear plan.

Hypothesis: branched plans can achieve costs competitive with those obtained by online replanning. To test our final hypothesis we consider an alternative definition of failure based on 'twig' execution semantics. We attach a branch of the empty plan to each plan node (we refer to such small branches as twigs), and calculate the weakest preconditions for continuing with the plan. This has the effect that execution is less likely to break C: execution terminates if it is not likely to succeed with $P \geq \theta$, and the cost achieved so far is reported; rather than attempting action execution and potentially violating the constraints.

This allows us to compare to an simulation of online replanning using a Monte-Carlo approach. First, we generate a plan (limited to 60s of CPU time) and then simulate the execution of the first step (for efficiency this same plan is used as the initial plan for every simulation run). From the resulting state S', we replan, limiting search to 10s. Then, we execute the first step of the plan from S'; and replan again; repeating until the planner returns an empty plan, or C is violated (signalling failure). As a point of efficiency, if the remainder of the plan from S' is still acceptable (executes with confidence θ), it serves as an incumbent solution for search, inducing a cost bound. Ultimately, a simulation run terminates when it fails, or there are no further actions to execute, whereupon the cost of this state is returned.

The average costs achieved by replanning are included in Table 1 as Cost(R) and those for the closest analogue Twig approach with the same T_i and T_b. Clearly, branched plans are at a disadvantage to replanning as the latter is equivalent to knowing the values for which to compute a branch at each point in the plan. However, it is pleasing to note that despite not knowing the actual execution-time state the improvement between the original plan and the branched plan is much greater than that between replanning and the branched plan, meaning much of the benefit of online planning can be gained through the use of branched plans. Indeed taking the geometric mean across the ratios Cost(B)/Cost(0) and Cost(R)/Cost(O) shows that whilst branching reduces cost to 64% of Cost(O), replanning is not substantially better, reducing the cost to 54% of Cost(O). This is very encouraging for the use of branching, especially where online replanning is infeasible.

The failure rates for twig plans, and replanning, are much lower than those under the original execution semantics: this is not surprising as the executive has a 'bail-out' option prior to each action. The twigged plans and replanning simulation have similar failure rates, as would be expected: each uses the same criterion to determine whether or not to continue execiting the plan.

5 Conclusions and Future Work

We have introduced a method for generating plans for over-subscription planning problems in the presence of uncertainty about numeric resources. We have shown how extra branches can be added to the plan for selection at execution time if conditions are favourable. We have demonstrated that the use of branched plans allows lower cost to be achieved upon execution whilst still adhering to strict safety conditions.

In the future we want to investigate whether further cost improvements can be made by making more intelligent decisions about where to build branches, and the labels to use. We wish to further investigate methods for compressing the representation of trees: our current plan trees still contain some redundancy. We also wish to explore the application of our techniques in different problem domains.

6 Acknowledgements

We would like to thank Sylvie Thièbaux and Derek Long for helpful discussions about this work. The work is funded by EPSRC Post-Doctoral Fellowship (EP/H029001/1).

References

Baier, J.; Bacchus, F.; and McIlraith, S. 2007. A heuristic search approach to planning with temporally extended preferences. In *IJCAI 2007*.

Beaudry, E.; Kabanza, F.; and Michaud, F. 2010. Planning witn concurrency under resources and time uncertainty. In *Proc. ECAI*.

Benton, J.; Do, M. B.; and Kambhampati, S. 2009. Anytime heuristic search for partial satisfaction planning. *AIJ*.

Coles, A. J., and Coles, A. I. 2011. LPRPG-P: Relaxed plan heuristics for planning with preferences. In *Proc. ICAPS*.

Conrad, P. R., and Williams, B. C. 2011. Drake: An Efficient Executive for Temporal Plans with Choice. *Journal of Artificial Intelligence Research (JAIR)* 42:607–659.

Drummond, M.; Bresina, J.; and Swanson, K. 1994. Just-in-Case Scheduling. In *Proc. AAAI*, 1098–1104.

Edelkamp, S., and Kissmann, P. 2008. GAMER: Bridging Planning and General Game Playing with Symbolic Search. In *IPC6 booklet, ICAPS*.

Edelkamp, S.; Jabbar, S.; and Nazih, M. 2006. Large-Scale Optimal PDDL3 Planning with MIPS-XXL. In *IPC5 booklet, ICAPS*.

Fox, M.; Long, D.; Baldwin, L.; Wilson, G.; Woods, M.; Jameux, D.; and Aylett, R. 2006. On-board timeline validation and repair: A feasibility study. In *Proc. IWPSS*.

Gerevini, A. E.; Long, D.; Haslum, P.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic Planning in the Fifth International Planning Competition: PDDL3 and Experimental Evaluation of the Planners. *AIJ*.

Hoffmann, J. 2003. The Metric-FF Planning System: Translating Ignoring Delete Lists to Numeric State Variables. *JAIR* 20.

Howey, R.; Long, D.; and Fox, M. 2004. Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In *Proc. ICTAI*.

Mausam, and Weld, D. S. 2008. Planning with Durative Actions in Stochastic Domains. *Journal of Artificial Intelligence Research (JAIR)* 31:38–82.

Meuleau, N.; Benazera, E.; Brafman, R. I.; Hansen, E. A.; and Mausam. 2009. A Heuristic Search Approach to Planning with Continuous Resources in Stochastic Domains. *JAIR* 34:27–59.

Rachelson, E.; Quesnel, G.; Garcia, F.; and Fabiani, P. 2008. A simulation-based approach for solving temporal markov problems. In *Proc. ECAI*.

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

Younes, H., and Simmons, R. G. 2004. Policy generation for continuous-time stochastic domains with concurrency. In *Proc. ICAPS*.

The Dynamic Controllability of Conditional STNs with Uncertainty

Luke Hunsberger*
hunsberg@cs.vassar.edu
Vassar College
Poughkeepsie, NY 12604

Roberto Posenato
roberto.posenato@univr.it
University of Verona
Verona, Italy

Carlo Combi
carlo.combi@univr.it
University of Verona
Verona, Italy

Abstract

Recent attempts to automate business processes and medical-treatment processes have uncovered the need for a formal framework that can accommodate not only temporal constraints, but also observations and actions with uncontrollable durations. To meet this need, this paper defines a Conditional Simple Temporal Network with Uncertainty (CSTNU) that combines the simple temporal constraints from a Simple Temporal Network (STN) with the conditional nodes from a Conditional Simple Temporal Problem (CSTP) and the contingent links from a Simple Temporal Network with Uncertainty (STNU). A notion of dynamic controllability for a CSTNU is defined that generalizes the dynamic consistency of a CTP and the dynamic controllability of an STNU. The paper also presents some sound constraint-propagation rules for dynamic controllability that are expected to form the backbone of a dynamic-controllability-checking algorithm for CSTNUs.

Introduction and Motivation

Workflow systems have been used to model business, manufacturing and medical-treatment processes. However, as Bettini et al. (2002) observed: "It would greatly enhance the capabilities of current workflow systems if quantitative temporal constraints on the duration of activities and their synchronization requirements can be specified and reasoned about." Toward that end, Combi et al. (2007; 2009; 2010) presented a new workflow model that accommodates the following key features: tasks with uncertain/uncontrollable durations; temporal constraints among tasks; and branching paths, where the branch taken is not known in advance. Fig. 1 shows a sample workflow from the health-care domain, similar to one presented by Combi and Posenato (2009). In this workflow, all times are in minutes, and:

- tasks are represented by rounded boxes;
- branching points are represented by nine-sided boxes called *split* or *join connectors*[1];
- tasks and connectors have duration attributes, $[x, y]$;

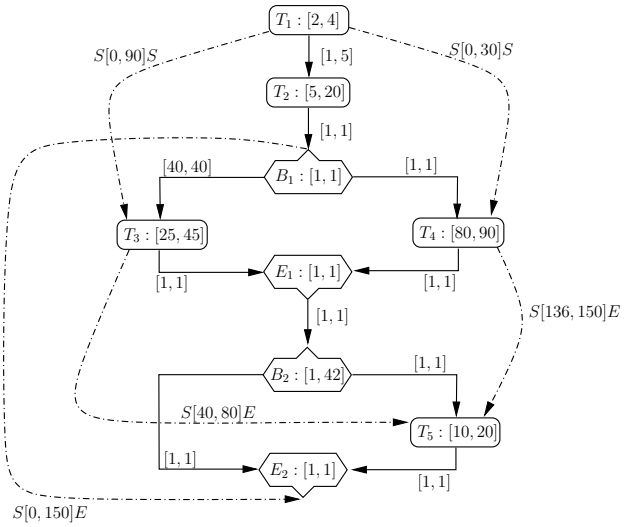

Figure 1: A sample workflow

- the flow, represented by solid arrows moving downward, specifies a partial order among tasks and connectors;
- intervals between consecutive tasks or connectors—called *delays*—are bounded by intervals of the form $[x, y]$;
- additional temporal constraints are represented by dashed arrows, also labeled by intervals of the form $[x, y]$.

The S and E notations on temporal constraints are used to indicate whether a constraint applies to the *starting* or *ending* times of the tasks/connectors it links. For example, the notation $S[136, 150]E$ on the arrow from T_4 to T_5 indicates that the duration of the interval from the *start* of T_4 to the *end* of T_5 must be in the range, $[136, 150]$.

The tasks and their uncontrollable durations are:

T_1: Pre-hospital issues, 2–4 min.

T_2: Initial patient evaluation, 5–20 min.

T_3: Percutaneous Coronary Intervention, 25–45 min.

T_4: Reperfusion Fibrinolytic therapy, 80–90 min.

T_5: Ancillary therapy, 10–20 min.

The semantics of execution for workflows stipulates that:

*Funded in part by the Phoebe H. Beadle Science Fund.

[1]Combi and Posenato (2009) used diamonds for connectors.

- The agent is free to choose starting times for all tasks, but does not control their durations; instead, the agent merely observes the durations of tasks in real time.
- The agent is free to choose starting and ending times for all connectors, but does not control which branch a *split* connector will follow; instead, the agent merely observes which branch is followed in real time.[2]

If a workflow admits a strategy for executing its tasks and connectors such that

- its execution decisions depend only on past observations of task durations and branch directions; and
- all delays and temporal constraints are guaranteed to be satisfied no matter how the task durations turn out and no matter which branches are taken,

then that workflow is said to be *history-dependent controllable* (Combi and Posenato 2009; 2010). The workflow in Fig. 1 is history-dependent controllable. A successful strategy must restrict the interval for B_2 to be $[32, 42]$ if the branch containing task T_4 was taken, or $[1, 31]$ if the branch containing task T_3 was taken. Combi and Posenato (2010) presented an exponential-time algorithm for determining whether any workflow is history-dependent controllable.

The rest of this paper introduces a Conditional Simple Temporal Network with Uncertainty (CSTNU) that provides a formal representation for the time-points and temporal constraints underlying workflows. A CSTNU is shown to generalize three existing kinds of temporal networks from the literature. Similarly, the concept of dynamic controllability for a CSTNU is shown to generalize analogous concepts for existing kinds of networks, while also being related to the history-dependent controllability for a workflow.

Related Temporal Networks

This section summarizes three kinds of temporal networks from the literature: Simple Temporal Networks, Conditional Simple Temporal Problems, and Simple Temporal Networks with Uncertainty. For convenience, we replace the Conditional Simple Temporal Problem with an equivalent alternative: the Conditional Simple Temporal Network.

Simple Temporal Networks

Definition 1 (STN)**.** (Dechter, Meiri, and Pearl 1991) A *Simple Temporal Network* (STN) is a pair, $(\mathcal{T}, \mathcal{C})$, where $\mathcal{T}$ is a set of real-valued variables, called *time-points,* and $\mathcal{C}$ is a set of binary constraints, called *simple temporal constraints.* Each constraint in $\mathcal{C}$ has the form, $Y - X \leq \delta$, where X and Y are any time-points in $\mathcal{T}$, and δ is any real number. A *solution* to the STN, $(\mathcal{T}, \mathcal{C})$, is a complete set of assignments to the variables in $\mathcal{T}$ that satisfy all of the constraints in $\mathcal{C}$.

Conditional Simple Temporal Networks

A Conditional Simple Temporal Problem (CSTP) augments an STN to include observation time-points (or observation nodes) (Tsamardinos, Vidal, and Pollack 2003). Each observation time-point has a proposition associated with it. When the observation time-point is executed, the truth-value of its associated proposition becomes known. In addition, each time-point has a *label* that restricts the scenarios in which that time-point can be executed. For example, a label, $A\neg B$, on a time-point would indicate that that time-point could only be executed in scenarios where the proposition A was true and B was false.

Although not included in the formal definition, the authors made the following *reasonability assumptions* about CSTPs:

(A1) A CSTP should not include any constraint relating time-points whose labels are inconsistent.

(A2) If the label on some time-point T includes a proposition Q, then the observation node, T_Q, associated with Q must be executed in all cases in which T is executed, and T_Q must be executed *before* T (i.e., $T_Q < T$).[3]

This section defines a Conditional Simple Temporal Network (CSTN), which is the same as a CSTP except that:

- the CSTN definition explicitly incorporates the reasonability assumptions (A1) and (A2) (cf. conditions WD1 and WD2 in Defn. 4, below); and
- each *constraint* in a CSTN has a label associated with it that subsumes the labels of the time-points it constrains (cf. conditions WD1 and WD3 in Defn. 4, below).

Putting labels on the constraints will facilitate the propagation of constraints, as is discussed later on.

Definition 2 (Label, Label Universe)**.** Given a set P of propositional letters, a *label* is any (possibly empty) conjunction of (positive or negative) literals from P. For convenience, the empty label is denoted by $\boxdot$. The *label universe* of P, denoted by P^*, is the set of all labels whose literals are drawn from P.

For example, if $P = \{A, B\}$, then

$P* = \{\boxdot, A, B, \neg A, \neg B, AB, A\neg B, \neg AB, \neg A\neg B\}$.

Definition 3 (Consistent labels, label subsumption)**.**

- Two labels, ℓ_1 and ℓ_2, are called *consistent*, denoted by $Con(\ell_1, \ell_2)$, if and only if $\ell_1 \wedge \ell_2$ is satisfiable.
- A label ℓ_1 *subsumes* a label ℓ_2, denoted by $Sub(\ell_1, \ell_2)$, if and only if $\models (\ell_1 \Rightarrow \ell_2)$.

To facilitate comparison with the definition of a CSTP, which is not repeated here due to space limitations, the order of arguments in a CSTN is the same as in a CSTP.

Definition 4 (CSTN)**.** A *Conditional Simple Temporal Network* (CSTN) is a tuple, $\langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$, where:

- $\mathcal{T}$ is a finite set of real-valued time-points;
- P is a finite set of propositional letters (or propositions);
- $L : \mathcal{T} \rightarrow P^*$ is a function that assigns a label to each time-point in $\mathcal{T}$;
- $\mathcal{OT} \subseteq \mathcal{T}$ is a (finite) set of observation time-points;
- $\mathcal{O} : P \rightarrow \mathcal{OT}$ is a bijection that associates a unique observation time-point to each propositional letter;

[2] This paper restricts attention to *conditional* connectors. Combi and Posenato (2009) discuss additional kinds of connectors.

[3] Simple temporal constraints do not allow for strict inequalities such as $Y < X$; however, in practice, a constraint such as $Y - X \leq -\epsilon$, for some small $\epsilon > 0$, achieves the desired effect.

- $\mathcal{C}$ is a set of *labeled* simple temporal constraints, each having the form, $(Y - X \leq \delta,\ \ell)$, where $X, Y \in \mathcal{T}$, δ is a real number, and $\ell \in P^*$;

(WD1) for any labeled constraint, $(Y - X \leq \delta,\ \ell) \in \mathcal{C}$, the label ℓ is satisfiable and subsumes both $L(X)$ and $L(Y)$;

(WD2) for each $p \in P$, and each $T \in \mathcal{T}$ for which either p or $\neg p$ appears in T's label,

- $Sub(L(T), L(\mathcal{O}(p)))$, and
- $(\mathcal{O}(p) - T \leq -\epsilon, L(T)) \in \mathcal{C}$, for some $\epsilon > 0$; and

(WD3) for each labeled constraint, $(Y - X \leq \delta,\ \ell)$, and for each $p \in P$ for which either p or $\neg p$ appears in ℓ,

- $Sub(\ell, L(\mathcal{O}(p)))$.

The following definitions will facilitate the proofs of the subsequent lemmas.

Definition 5 ($\mathcal{C}_\boxdot$, $L^{\mathcal{T}}_\boxdot$ and $\mathcal{O}_\emptyset$)**.**

- If $\mathcal{C}$ is a set of simple temporal constraints, then $\mathcal{C}_\boxdot$ is the corresponding set of *labeled* simple temporal constraints, where each constraint is labeled by the empty label, $\boxdot$. In particular, $\mathcal{C}_\boxdot = \{(Y - X \leq \delta,\ \boxdot) \mid (Y - X \leq \delta) \in \mathcal{C}\}$
- For any set $\mathcal{T}$ of time-points, $L^{\mathcal{T}}_\boxdot$ denotes the labeling function that assigns the empty label to each time-point. Thus, $L^{\mathcal{T}}_\boxdot(T) = \boxdot$ for all $T \in \mathcal{T}$. When the context allows, we may write $L_\boxdot$ instead of $L^{\mathcal{T}}_\boxdot$.
- $\mathcal{O}_\emptyset$ denotes the unique function whose domain and range are both empty. Thus, $\mathcal{O}_\emptyset : \emptyset \rightarrow \emptyset$.

The following lemmas show that any STN or CSTP can be embedded within a CSTN.

Lemma 1. *Let $(\mathcal{T}, \mathcal{C})$ be any STN. Then $\langle \mathcal{T}, \mathcal{C}_\boxdot, L_\boxdot, \emptyset, \mathcal{O}_\emptyset, \emptyset \rangle$ is a CSTN.*

Proof. We need only check that the conditions WD1, WD2 and WD3 from the definition of a CSTN are satisfied. WD2 and WD3 are trivially satsified since $P = \emptyset$. As for WD1, each constraint in $\mathcal{C}_\boxdot$ has $\boxdot$ as its label, which is satisfiable. Furthermore, $L_\boxdot$ assigns the empty label to every node. Thus, the empty label on each constraint trivially subsumes the empty label on the relevant nodes. □

Lemma 2 (Any CSTP is a CSTN)**.** *Let $\langle V, E, L, OV, \mathcal{O}, P \rangle$ be any CSTP, as defined by Tsamardinos et al. (2003), that satisfies the reasonability assumptions, A1 and A2. Let $\mathcal{S} = \langle V, \mathcal{C}, L, OV, \mathcal{O}, P \rangle$, where:*[4]

$$\mathcal{C} = \bigcup_{(a \leq Y - X \leq b) \in E} \{(a \leq Y - X \leq b, L(X) \wedge L(Y))\}$$

Then $\mathcal{S}$ is a CSTN.

Proof. Conditions WD1, WD2 and WD3 in the definition of a CSTN (Defn. 4) are satisfied as follows.

(WD1) Each labeled constraint in $\mathcal{C}$ has the form, $(V - U \leq \delta,\ L(U) \wedge L(V))$. Note that $L(U) \wedge L(V)$ subsumes both $L(U)$ and $L(V)$. Furthermore, by assumption A1, $L(U)$ and $L(V)$ must be mutually satisfiable.

(WD2) WD2 is simply a restatement of assumption A2.

(WD3) Each constraint in $\mathcal{C}$ has the form, $(V - U \leq \delta, \ell)$, where $\ell = L(U) \wedge L(V)$. By WD2, $L(U)$ must subsume $L(\mathcal{O}(p))$. But then ℓ does too. □

Simple Temporal Networks with Uncertainty

A Simple Temporal Network with Uncertainty (STNU) augments an STN to include a set, $\mathcal{L}$, of contingent links (Morris, Muscettola, and Vidal 2001). Each contingent link has the form, (A, x, y, C), where A and C are time-points, and $0 < x < y < \infty$. A is called the *activation* time-point; C is called the *contingent* time-point. An agent typically activates a contingent link by executing A. After doing so, the execution of C is out of the agent's control; however, C is guaranteed to execute such that the temporal difference, $C - A$, is between x and y. Contingent links are used to represent actions with uncertain durations; the agent initiates the action, but then merely *observes* its completion in real time.

Definition 6 (STNU)**.** A Simple Temporal Network with Uncertainty (STNU) is a triple, $(\mathcal{T}, \mathcal{C}, \mathcal{L})$, where:

- $(\mathcal{T}, \mathcal{C})$ is an STN; and
- $\mathcal{L}$ is a set of contingent links, each having the form, (A, x, y, C), where A and C are distinct time-points in $\mathcal{T}$, $0 < x < y < \infty$, and:
  - for each $(A, x, y, C) \in \mathcal{L}$, $\mathcal{C}$ contains the constraints, $(x \leq C - A \leq y)$ (cf. Footnote 4);
  - if (A_1, x_1, y_1, C_1) and (A_2, x_2, y_2, C_2) are distinct contingent links in $\mathcal{L}$, then C_1 and C_2 are distinct time-points; and
  - the contingent time-point for one contingent link may serve as the activation time-point for another—thus, contingent links may form chains or trees—however, contingent links may not form loops.

As will be seen, the semantics for contingent links is built into the definition of dynamic controllability.

Note that if $(\mathcal{T}, \mathcal{C})$ is an STN, then $(\mathcal{T}, \mathcal{C}, \emptyset)$ is an STNU.

Conditional STNs with Uncertainty

This section introduces a Conditional STN with Uncertainty (CSTNU) which combines features of CSTNs and STNUs.

Definition 7 ($\lfloor \mathcal{C} \rfloor$)**.** If $\mathcal{C}$ is a set of labeled constraints of the form, $(Y - X \leq \delta, \ell)$, then $\lfloor \mathcal{C} \rfloor$ is the corresponding set of *unlabeled* constraints:

$$\lfloor \mathcal{C} \rfloor = \{(Y - X \leq \delta) \mid (Y - X \leq \delta, \ell) \in \mathcal{C} \text{ for some } \ell\}.$$

Definition 8 (CSTNU)**.** A Conditional STN with Uncertainty (CSTNU) is a tuple, $\langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P, \mathcal{L} \rangle$, where:

- $\langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$ is a CSTN;
- $(\mathcal{T}, \lfloor \mathcal{C} \rfloor, \mathcal{L})$ is an STNU; and
- for each $(A, x, y, C) \in \mathcal{L}$, $L(A) = L(C)$, and $\mathcal{C}$ contains the labeled constraints, $(x \leq C - A \leq y, L(A))$.[5]

The following lemmas show that any STNU or CSTN can be embedded within a CSTNU.

Lemma 3. *If $(\mathcal{T}, \mathcal{C}, \mathcal{L})$ is an STNU, then $\langle \mathcal{T}, \mathcal{C}_\boxdot, L_\boxdot, \emptyset, \mathcal{O}_\emptyset, \emptyset, \mathcal{L} \rangle$ is a CSTNU.*

[4] For convenience, we use the expression, $a \leq Y - X \leq b$, to represent the pair of constraints, $Y - X \leq b$ and $X - Y \leq -a$.

[5] $(x \leq C - A \leq y, L(A))$ is shorthand for the pair of labeled constraints, $(A - C \leq -x,\ L(A))$ and $(C - A \leq y, L(A))$.

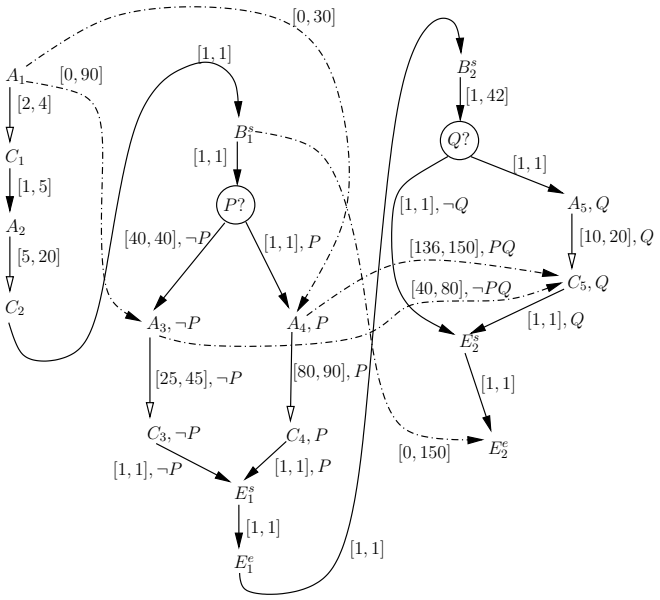

Figure 2: The CSTNU for the workflow in Fig. 1

Proof. Let $(\mathcal{T}, \mathcal{C}, \mathcal{L})$ be any STNU. Then $(\mathcal{T}, \mathcal{C})$ is an STN. By Lemma 1, $\langle \mathcal{T}, \mathcal{C}_\boxdot, L_\boxdot, \emptyset, \mathcal{O}_\emptyset, \emptyset \rangle$ is a CSTN. In addition, since $\lfloor \mathcal{C}_\boxdot \rfloor$ is necessarily the same as $\mathcal{C}$, $(\mathcal{T}, \lfloor \mathcal{C}_\boxdot \rfloor, \mathcal{L})$ must be an STNU. Finally, for each $(A, x, y, C) \in \mathcal{L}$, $\mathcal{C}$ contains the constraints, $(x \leq C - A \leq y)$, which implies that $\mathcal{C}_\boxdot$ contains the labeled constraints, $(x \leq C - A \leq y, \boxdot)$. Since $L_\boxdot$ assigns the empty label to each node, the last condition of Defn. 8 is satisfied. □

Lemma 4. *If $\langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$ is an CSTN, then $\langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P, \emptyset \rangle$ is a CSTNU.*

Proof. Let $\langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$ be any CSTN. Then $(\mathcal{T}, \lfloor \mathcal{C} \rfloor)$ is an STN, whence $(\mathcal{T}, \lfloor \mathcal{C} \rfloor, \emptyset)$ is an STNU. Since $\mathcal{L}$ is empty, the last condition of Defn. 8 is satisfied. □

The CSTNU Underlying a Worklow

Recall the sample workflow from Fig. 1. This workflow has an underlying CSTNU that is derived by

- replacing each task with a corresponding contingent link;
- replacing each split connector by a pair of (starting and ending) time-points, linked by a duration constraint, and where the ending time-point is an observation node for a proposition whose possible values correspond to the different branching decisions; and
- replacing each join connector by a pair of time-points, linked by a duration constraint.

Fig. 2 shows the CSTNU underlying the workflow from Fig. 1. In this CSTNU, each contingent link from A_i to C_i corresponds to the task T_i from the workflow; and observation nodes are circled. Note that the branch containing task T_4 is labeled by P, whereas the alternative branch containing task T_3 is labeled by $\neg P$. Similarly, the branch containing task T_5 is labeled by Q, and the alternative branch is labeled by $\neg Q$. Note, too, that labels on edges subsume the labels on the time-points they connect. Dashed edges are kept dashed to facilitate comparison with the workflow in Fig. 1.

Dynamic Controllability

This section combines the semantics of CSTNs and STNUs to generate a definition for the dynamic controllability of a CSTNU. Because the semantics for the corresponding notions involve similar definitions, in some cases the various terms, such as *history* or *dynamic* will be given prefixes or superscripts to indicate the kinds of networks or situations/scenarios they apply to. In addition, we use the term, *history*, instead of *pre-history*, for convenience.

Dynamic Consistency of CSTNs

A CSTP is called *dynamically consistent* if there exists a strategy for executing its time-points that guarantees the satisfaction of all relevant constraints no matter how the truth values of the various observations turn out (Tsamardinos, Vidal, and Pollack 2003). The strategy is dynamic in that its execution decisions can react to past observations, but not those in the future. This section defines the dynamic consistency of a CSTN in an equivalent way; however, for convenience, there are some superficial differences in notation and organization. Afterward, we provide a second characterization of the *dynamic* property that will be useful later on.

Definition 9 (Scenario/Interpretation Function)**.** A *scenario* (or *interpretation function*) over a set P of propositional letters is a function, $s : P \rightarrow \{true, false\}$, that assigns a truth value to each letter in P.[6] As is standard practice in propositional logic, any interpretation function can be extended to provide the truth value for every possible formula involving the letters in P. Thus, any interpretation function, s, can provide the truth value of each label involving letters in P. For any label, ℓ, the truth value of ℓ in the scenario, s, is denoted by $s(\ell)$. Let $\mathcal{I}_P$ (or simply $\mathcal{I}$) denote the set of all interpretation functions (or complete execution scenarios) over P.

Definition 10 (Schedule)**.** A *schedule* for a set of time-points $\mathcal{T}$ is a mapping, $\psi : \mathcal{T} \rightarrow \mathbb{R}$ that assigns a real number to each time-point in $\mathcal{T}$. The set of all schedules *for any subset of* $\mathcal{T}$ is denoted by $\Psi_\mathcal{T}$ (or Ψ if the context allows).

Below, the projection of a CSTN, $\mathcal{S}$, onto a scenario, s, is defined to be the STN that contains all of the time-points and constraints from $\mathcal{S}$ whose labels are true under s (i.e., the time-points that must be executed under s, and the constraints that must be satisfied under s).

Definition 11 (Scenario Projection for a CSTN)**.** Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$ be any CSTN, and s any interpretation function (i.e., complete scenario) for the letters in P. The *projection* of $\mathcal{S}$ onto the scenario s—denoted by $scPrj(\mathcal{S}, s)$—is the STN, $(\mathcal{T}_s^+, \mathcal{C}_s^+)$, where:

- $\mathcal{T}_s^+ = \{T \in \mathcal{T} : s(L(T)) = true\}$; and
- $\mathcal{C}_s^+ = \{(Y - X \leq \delta) \mid$ for some ℓ, $(Y - X \leq \delta, \ell) \in \mathcal{C}$ and $s(\ell) = true\}$

[6] Unlike the prior work on CSTPs, we restrict attention to *complete* scenarios because the subsequent definition of a *history* requires a scenario to entail the outcome of *all* past observations.

Recall that condition WD1 from the definition of a CSTN stipulates that the label on any constraint must subsume the labels on the time-points it connects. Thus, for any constraint in $\mathcal{C}_s^+$, the time-points it connects must belong to the set $\mathcal{T}_s^+$.

Definition 12 (Execution Strategy for a CSTN)**.** Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$ be any CSTN. An *execution strategy* for $\mathcal{S}$ is a mapping, $\sigma : \mathcal{I} \rightarrow \Psi_{\mathcal{T}}$, such that for each scenario, $s \in \mathcal{I}$, the domain of $\sigma(s)$ is $\mathcal{T}_s^+$ (cf. Defn. 11). If, in addition, for each scenario, s, the schedule $\sigma(s)$ is a solution to the scenario projection, $scPrj(\mathcal{S}, s)$, then σ is called *viable.* In any case, the execution time for the time-point X in the schedule $\sigma(s)$ is denoted by $[\sigma(s)]_X$.

Below, the *history* of a time-point, X, relative to a scenario, s, and strategy, σ, is defined to be the set of observations made before the time at which X is executed according to the schedule, $\sigma(s)$ (i.e., before the time $[\sigma(s)]_X$).[7]

Definition 13 (Scenario history for a CSTN)**.** Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$ be any CSTN, s any scenario, σ any execution strategy for $\mathcal{S}$, and X any time-point in $\mathcal{T}_s^+$ (cf. Defn. 11). The *history* of X in the scenario s, for the strategy σ—denoted by $scHst(X, s, \sigma)$—is given by:

$$scHst(X, s, \sigma) = \{(p, s(p)) \mid \mathcal{O}(p) \in \mathcal{T}_s^+, \text{ and } [\sigma(s)]_{\mathcal{O}(p)} < [\sigma(s)]_X\}$$

Note that any scenario history determines a corresponding label whose (positive or negative) literals are in a one-to-one correspondence with the observations, $(p, s(p))$, in the history. Thus, we may sometimes (e.g., in the next definition) treat a scenario history as though it were a label.

Below, an execution strategy is called *dynamic* if the schedules it generates always assign the same execution time to any time-point X in scenarios that cannot be distinguished prior to that time.[8]

Definition 14 (Dynamic Execution Strategy for a CSTN)**.** An execution strategy, σ, for a CSTN is called *dynamic* if for all scenarios, s_1 and s_2, and any time-point X:

if $Con(s_1, scHst(X, s_2, \sigma))$, then $[\sigma(s_1)]_X = [\sigma(s_2)]_X$.

Definition 15 (Dynamic Consistency for a CSTN)**.** A CSTN is called *dynamically consistent* if there exists an execution strategy for it that is both viable and dynamic.

The following definitions and lemma provide an equivalent, alternative characterization of a dynamic execution strategy for a CSTN. First, a scenario history relative to a numerical time—not a time-point variable—is defined.

Definition 16 (Scenario History* for a CSTN)**.** Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$ be any CSTN, s any scenario, σ any execution strategy for $\mathcal{S}$, and t any real number. The *history** of t in the scenario s, for the strategy σ—denoted by $scHst^*(t, s, \sigma)$—is the set of all observations made before time t according to the schedule, $\sigma(s)$. In particular:

$$scHst^*(t, s, \sigma) = \{(p, s(p)) \mid \mathcal{O}(p) \in \mathcal{T}_s^+ \text{ and } [\sigma(s)]_{\mathcal{O}(p)} < t\}$$

Note that for all time-points X, scenarios s, and strategies σ, $scHst(X, s, \sigma) = scHst^*([\sigma(s)]_X, s, \sigma)$.

Definition 17 (Dynamic* Execution Strategy for a CSTN)**.** An execution strategy, σ, for an CSTN is called *dynamic** if for any scenarios, s_1 and s_2, and any time-point, X:

if $scHst^*([\sigma(s_1)]_X, s_1, \sigma) = scHst^*([\sigma(s_1)]_X, s_2, \sigma)$, then $[\sigma(s_1)]_X = [\sigma(s_2)]_X$.

Notice that in this definition, the two histories, one relative to s_1, the other to s_2, are taken with respect to the *same* (numeric) time, $[\sigma(s_1)]_X$. If the strategy σ yields schedules for s_1 and s_2 that have identical histories prior to that one time, then those schedules must assign the same value to X.

Lemma 5. *An execution strategy σ for a CSTN is dynamic if and only if it is dynamic**.

Proof.

($\Rightarrow$) Suppose σ is a dynamic execution strategy for some CSTN. Let s_1 and s_2 be any scenarios, and X any time-point such that $scHst^*(t_1, s_1, \sigma) = scHst^*(t_1, s_2, \sigma)$, where $t_1 = [\sigma(s_1)]_X$. Now s_2 must be consistent with $scHst^*(t_1, s_2, \sigma)$, since the observations contained in that history are a subset of the observations in s_2. Thus, s_2 is consistent with $scHst^*(t_1, s_1, \sigma)$, which equals $scHst(X, s_1, \sigma)$. Thus, since σ is dynamic, we must have that $[\sigma(s_1)]_X = [\sigma(s_2)]_X$. Thus, σ is dynamic*.

($\Leftarrow$) Suppose σ is a dynamic* execution strategy for some CSTN. Let s_1 and s_2 be any scenarios, and X any time-point such that $Con(s_1, scHist(X, s_2, \sigma))$. Suppose that $[\sigma(s_1)]_X \neq [\sigma(s_2)]_X$. Let $t \in \mathbb{R}$ be the first time at which the schedules $\sigma(s_1)$ and $\sigma(s_2)$ diverge. Then, $t \leq \min\{[\sigma(s_1)]_X, [\sigma(s_2)]_X\}$; and there must be some time-point Y that is executed at time t in one scenario, and at some later time in the other scenario.

By construction, $t \leq [\sigma(s_2)]_X$. Thus, $scHst^*(t, s_2, \sigma) \subseteq scHst^*([\sigma(s_2)]_X, s_2, \sigma) = scHst(X, s_2, \sigma)$. Thus, since $Con(s_1, scHst(X, s_2, \sigma))$, it follows that $Con(s_1, scHst^*(t, s_2, \sigma))$. And since s_1 is a complete scenario, the observations in $scHst^*(t, s_2, \sigma)$ must be a subset of the "observations" in s_1. And since, by construction, the schedules, $\sigma(s_1)$ and $\sigma(s_2)$, are identical prior to time t, it follows that the observations in the two histories, $scHst^*(t, s_1, \sigma)$ and $scHst^*(t, s_2, \sigma)$, involve the same sets of observation time-points with identical outcomes (i.e., truth values). Thus, $scHst^*(t, s_1, \sigma) = scHst^*(t, s_2, \sigma)$, whence the property of σ being dynamic* implies that $[\sigma(s_1)]_Y = [\sigma(s_2)]_Y$, contradicting the choice of Y. Thus, it must be that the schedules, $\sigma(s_1)$ and $\sigma(s_2)$ diverge, if at all, *after* the execution of X, in which case, $[\sigma(s_1)]_X = [\sigma(s_2)]_X$. Thus, σ is dynamic. □

Dynamic Controllability of STNUs

Morris et al. (2001) call an STNU *dynamically controllable* if there exists a strategy for executing its time-points that

[7] Tsamardinos et al. (2003) define (pre)histories for arbitrary schedules, whereas here we restrict attention to schedules of the form, $\sigma(s)$, where σ is an execution strategy and s is a scenario.

[8] Tsamardinos et al. (2003) include a disjunctive condition, $Con(s_1, scHst(X, s_2, \sigma)) \vee Con(s_2, scHist(X, s_1, \sigma))$. However, since s_1 and s_2 play symmetric roles in the two disjuncts, and since s_1 and s_2 are both universally quantified (cf. Defn. 14), it suffices to include just one of the disjuncts.

guarantees the satisfaction of all constraints in the network no matter how the durations of the contingent links turn out. The strategy is dynamic in that its execution decisions can react to observations of contingent links that have already completed, but not to those that have yet to complete.

This section presents a sequence of definitions that culminate in the definition of the dynamic controllability of an STNU. Most of the definitions are from Morris et al. (2001), albeit with some slight differences in notation, but *history** and *dynamic** are from Hunsberger (2009). Parallels between the definitions in this section and those from the preceding section are highlighted along the way.

Analogous to a scenario for a CSTN, which specifies the truth value for each proposition, a *situation* for an STNU specifies fixed durations for all of the contingent links.

Definition 18 (Situations)**.** Let $\mathcal{S}$ be an STNU having the k contingent links, $(A_1, x_1, y_1, C_1), \ldots, (A_k, x_k, y_k, C_k)$, with respective duration ranges, $[x_1, y_1], \ldots, [x_k, y_k]$. Then $\Omega_{\mathcal{S}} = [x_1, y_1] \times \ldots \times [x_k, y_k]$ is called the *space of situations* for $\mathcal{S}$. Any $\omega = (d_1, \ldots, d_k) \in \Omega_{\mathcal{S}}$ is called a *situation.* When context allows, we may write Ω instead of $\Omega_{\mathcal{S}}$.

Schedules for STNUs are defined the same way as for CSTNs, except that the domain for each schedule must be the entire set of time-points, $\mathcal{T}$.

The projection of a CSTN onto a scenario yields an STN by fixing the truth value of each propositional letter and restricting attention to those time-points and constraints whose labels are true according to that scenario. Analogously, the projection of an STNU onto a situation yields an STN by fixing the duration of each contingent link.

Definition 19 (Situation Projection for an STNU)**.** Suppose $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ is an STNU and $\omega = (d_1, \ldots, d_k)$ is a situation. The *projection* of $\mathcal{S}$ onto the situation ω—denoted by $sitPrj(\mathcal{S}, \omega)$—is the STN, $(\mathcal{T}, \mathcal{C}'')$, where:

$$\mathcal{C}'' \;=\; \mathcal{C} \;\cup\; \{(d_i \leq C_i - A_i \leq d_i) \mid 1 \leq i \leq k\}.$$

Definition 20 (Execution Strategy for an STNU)**.** Let $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ be an STNU. An *execution strategy* for $\mathcal{S}$ is a mapping, $\sigma : \Omega \rightarrow \Psi$, such that for each situation, $\omega \in \Omega$, $\sigma(\omega)$ is a (complete) schedule for the time-points in $\mathcal{T}$. If, in addition, for each situation, ω, the schedule $\sigma(\omega)$ is a solution for the situation projection, $sitPrj(\mathcal{S}, \omega)$, then σ is called *viable.* In any case, the execution time for any time-point X in the schedule, $\sigma(\omega)$, is denoted by $[\sigma(\omega)]_X$.

Analogous to a scenario history* for a CSTN, a situation history* for an STNU specifies the durations of all contingent links that have finished executing prior to a (numeric) time t in a schedule $\sigma(\omega)$.

Definition 21 (Situation History* for an STNU)**.** Let $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ be any STNU, σ any execution strategy for $\mathcal{S}$, ω any situation, and t any real number. The *history** of t in the situation ω, for the strategy σ—denoted by $sitHst(t, \omega, \sigma)$—is the set:

$$sitHst(t, \omega, \sigma) \;=\; \{(A, C, [\sigma(\omega)]_C - [\sigma(\omega)]_A) \mid \exists x, y \text{ s.t. } (A, x, y, C) \in \mathcal{L} \text{ and } [\sigma(\omega)]_C < t\}$$

The definition of the *dynamic** property for an execution strategy for an STNU parallels that of the *dynamic** property for an execution strategy for a CSTN.

Definition 22 (Dynamic* Execution Strategy for an STNU)**.** An execution strategy, σ, for an STNU is called *dynamic** if for any situations, ω_1 and ω_2, and any *non-contingent* time-point X:

if $sitHst([\sigma(\omega_1)]_X, \omega_1, \sigma) = sitHst([\sigma(\omega_1)]_X, \omega_2, \sigma)$,
then $[\sigma(\omega_1)]_X = [\sigma(\omega_2)]_X$.

Definition 23 (Dynamic Controllability for an STNU)**.** An STNU $\mathcal{S}$ is called *dynamically controllable* if there exists an execution strategy for $\mathcal{S}$ that is both viable and dynamic*.

Dynamic Controllability of CSTNUs

This section extends the notions of the dynamic consistency of a CSTN and the dynamic controllability of an STNU to generate a (novel) definition of the dynamic controllability of a CSTNU. To wit, a sequence of definitions is presented that parallels those of the preceding sections.

A *drama* is a scenario/situation pair that specifies fixed truth values for all of the propositional letters and fixed durations for all of the contingent links.

Definition 24 (Drama)**.** Given a CSTNU $\mathcal{S}$, a *drama* is any pair (s, ω), where s is a scenario, and ω is a situation. The set of all dramas (for $\mathcal{S}$) is $\mathcal{I} \times \Omega$.

Next, the *projection* of a CSTNU onto a drama, (s, ω), is defined. The projection restricts attention to time-points and constraints whose labels are true under the scenario s, while also including constraints that force the contingent links to take on the durations specified in the situation ω.

Definition 25 (Drama Projection for a CSTNU)**.** Suppose $\mathcal{S} = \langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P, \mathcal{L} \rangle$ is a CSTNU and (s, ω) is a drama for $\mathcal{S}$, where $\omega = (d_1, \ldots, d_k)$. The *projection* of $\mathcal{S}$ onto the drama (s, ω)—denoted by $drPrj(\mathcal{S}, s, \omega)$—is the STN, $(\mathcal{T}_s^+, \mathcal{C}_1 \cup \mathcal{C}_0)$, where:

- $\mathcal{T}_s^+ = \{T \in \mathcal{T} : s(L(T)) = true\}$
- $\mathcal{C}_1 = \{(Y - X \leq \delta) \mid$ for some ℓ, $(Y - X \leq \delta, \ell) \in \mathcal{C}$, and $s(\ell) = true\}$
- $\mathcal{C}_0 = \{(d_i \leq C_i - A_i \leq d_i) \mid (A_i, x_i, y_i, C_i) \in \mathcal{L}$ and $A_i, C_i \in \mathcal{T}_s^+\}$

Definition 26 (Execution Strategy for a CSTNU)**.** Let $\mathcal{S} = \langle \mathcal{T}, P, L, \mathcal{OT}, \mathcal{O}, \mathcal{C}, \mathcal{L} \rangle$ be a CSTNU. An *execution strategy* for $\mathcal{S}$ is a mapping, $\sigma : (\mathcal{I} \times \Omega) \rightarrow \Psi_{\mathcal{T}}$, such that for each drama, (s, ω), the domain of $\sigma(s, \omega)$ is $\mathcal{T}_s^+$. σ is called *viable* if for each drama, (s, ω), the schedule $\sigma(s, \omega)$ is a solution to the projection, $drPrj(s, \omega)$. For any time-point X and drama (s, ω), the execution time of X in the schedule, $\sigma(s, \omega)$, is denoted by $[\sigma(s, \omega)]_X$.

The following definition combines the definitions of history* relative to a numeric time for CSTNs and STNUs.

Definition 27 (Drama History* for a CSTNU)**.** Let $\mathcal{S} = \langle \mathcal{T}, P, L, \mathcal{OT}, \mathcal{O}, \mathcal{C}, \mathcal{L} \rangle$ be a CSTNU. Let σ be an execution strategy for $\mathcal{S}$, (s, ω) some drama, and t some real number. Then the *history** of t for the drama (s, ω) and strategy σ—denoted by $drHst(t, s, \omega, \sigma)$—is the pair $(\mathcal{H}_s, \mathcal{H}_\omega)$ where:

- $\mathcal{H}_s = \{(p, s(p)) \mid \mathcal{O}(p) \in \mathcal{T}_s^+$ and $[\sigma(s, \omega)]_{\mathcal{O}(p)} < t\}$; and

- $\mathcal{H}_\omega = \{(A, C, [\sigma(s,\omega)]_C - [\sigma(s,\omega)]_A) \mid A, C \in \mathcal{T}_s^+,$ $\exists x, y$ s.t. $(A, x, y, C,) \in \mathcal{L},\ [\sigma(s,\omega)]_C < t\}$.

Note that $\mathcal{H}_s$ specifies the truth values of all propositions that are observed prior to t in the schedule $\sigma(s,\omega)$; and $\mathcal{H}_\omega$ specifies the durations of all contingent links that finish executing prior to t in that schedule.

Definition 28 (Dynamic* Execution Strategy for a CSTNU)**.** An execution strategy, σ, for a CSTNU is called *dynamic** if for every pair of dramas, (s_1, ω_1) and (s_2, ω_2), and every *non-contingent* time-point $X \in \mathcal{T}_{s_1}^+ \cap \mathcal{T}_{s_2}^+$:

if $drHst(t, s_1, \omega_1, \sigma) = drHst(t, s_2, \omega_2, \sigma)$,
where $t = [\sigma(s_1, \omega_1)]_X$,
then $[\sigma(s_1, \omega_1)]_X = [\sigma(s_2, \omega_2)]_X$.

Definition 29 (Dynamic Controllability for a CSTNU)**.** A CSTNU, $\mathcal{S}$, is *dynamically controllable* if there exists an execution strategy for $\mathcal{S}$ that is both viable and dynamic*.

The following lemmas show that the above definition properly generalizes the dynamic consistency of a CSTN and the dynamic controllability of an STNU.

Lemma 6. *Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P\rangle$ be any CSTN. Then $\mathcal{S}$ is dynamically consistent if and only if the CSTNU, $\mathcal{S}_u = \langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P, \emptyset\rangle$, is dynamically controllable.*

Proof. Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P\rangle$ be any dynamically consistent CSTN. Then $\mathcal{S}$ has an execution strategy, $\sigma : \mathcal{I} \rightarrow \Psi_\mathcal{T}$, that is both viable and dynamic. By Lemma 5, σ is also dynamic*. In addition, since $\mathcal{S}$ is a CSTN, Lemma 4 ensures that $\mathcal{S}_u = \langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P, \emptyset\rangle$ is a CSTNU. We must show that $\mathcal{S}_u$ has an execution strategy, $\sigma_u : (\mathcal{I} \times \Omega) \rightarrow \Psi_\mathcal{T}$, that is both viable and dynamic*. Note that since $\mathcal{S}_u$ has no contingent links, Ω contains exactly one situation—the null situation—which we shall denote by $\omega_\emptyset$.

Define σ_u as follows. For any drama, $(s, \omega_\emptyset)$, let $\sigma_u(s, \omega_\emptyset) = \sigma(s)$. Note that σ_u is an execution strategy for $\mathcal{S}_u$, since the domain of $\sigma(s)$ is guaranteed to be $\mathcal{T}_s^+$.

Since σ is viable, for any scenario s, the schedule $\sigma(s)$ is a solution to the scenario projection, $scPrj(\mathcal{S}, s)$. However, for any s, the schedules, $\sigma(s)$ and $\sigma(s, \omega_\emptyset)$ are defined to be the same. Furthermore, since $\mathcal{S}_u$ has no contingent links, it follows that for any s, the drama projection, $drPrj(\mathcal{S}_u, s, \omega_\emptyset)$, is the same STN as $scPrj(\mathcal{S}, s)$ (cf. Defns. 11 and 25). Thus, for any s, $\sigma(s, \omega_\emptyset)$ is necessarily a solution to $drPrj(\mathcal{S}_u, s, \omega_\emptyset)$, whence σ_u is viable.

To show that σ_u is dynamic*, suppose $(s_1, \omega_\emptyset)$ and $(s_2, \omega_\emptyset)$ are any dramas in $\mathcal{I} \times \Omega$, X is a non-contingent time-point in $\mathcal{T}_{s_1}^+ \cap \mathcal{T}_{s_2}^+$, $t = [\sigma_u(s_1, \omega_\emptyset)]_X$, and $drHst^*(t, s_1, \omega_\emptyset, \sigma_u) = drHst^*(t, s_2, \omega_\emptyset, \sigma_u)$. Note that $t = [\sigma_u(s_1, \omega_\emptyset)]_X = [\sigma(s_1)]_X$. Furthermore, since there are no contingent links, $drHst^*(t, s_1, \omega_\emptyset, \sigma_u) = drHst^*(t, s_2, \omega_\emptyset, \sigma_u)$ if and only if $scHst^*(t, s_1, \sigma) = scHst^*(t, s_2, \sigma)$ (cf. Defns. 16 and 27). But then σ being dynamic* ensures that $[\sigma(s_1)]_X = [\sigma(s_2)]_X$ (cf. Defn. 17), and hence $[\sigma_u(s_1, \omega_\emptyset)]_X = [\sigma_u(s_2, \omega_\emptyset)]_X$. □

Lemma 7. *Let $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ be any STNU. Then $\mathcal{S}$ is dynamically controllable if and only if the CSTNU, $\langle \mathcal{T}, \mathcal{C}_\boxdot, L_\boxdot, \emptyset, \mathcal{O}_\emptyset, \emptyset, \mathcal{L}\rangle$, is dynamically controllable.*

Proof. The proof is omitted for space reasons. It has the same general structure as the proof of Lemma 6.

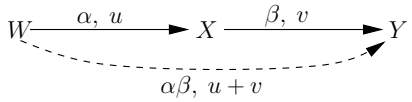

Figure 3: Basic constraint propagation in a CSTNU

Toward a DC-Checking Algorithm for CSTNUs

This section addresses the problem of finding an algorithm for determining the dynamic controllability of arbitrary CSTNUs. Given that CSTNUs combine the features of CSTPs and STNUs, one approach would be to combine existing algorithms for determining the dynamic consistency of CSTPs and the dynamic controllability of STNUs. However, those algorithms employ very different techniques. For example, in the CSTP algorithm, Tsamardinos et al. (2003) first derive a related Disjunctive Temporal Problem (DTP), and then solve it using a dedicated DTP solver that is optimized by a variety of constraint-satisfaction heuristics. In contrast, the fastest algorithm for determining whether arbitrary STNUs are dynamically controllable is the $O(N^4)$-time algorithm developed by Morris (2006), which is a constraint-propagation algorithm that focuses on the *reducing away of lower-case edges* in an STNU graph.

Another problem is that the CSTP algorithm uses exponential space and time. Conrad and colleagues (Conrad 2010; Conrad and Williams 2011) developed the Drake system for propagating *labeled* constraints in temporal networks with choice.[9] The aim was to reduce the space required to generate dispatchable plans, while accepting slight increases in the time requirements. Although their choice nodes are dramatically different from the observation nodes in a CSTNU—because choice nodes are *controlled* by the agent—their use of *labeled value sets* in constraint propagation inspired our use of labels on the edges of a CSTNU.

Constraint Propagation for CSTNUs

Consider the propagation of labeled constraints illustrated in Fig. 3. Any dynamic execution strategy that observes the labeled constraints from W to X, and from X to Y, must also observe the derived constraint from W to Y. Notice that the label on the derived constraint is the conjunction of the labels on the original constraints. The proof that this propagation rule is sound is omitted, due to space limitations.

Label Modification in a CSTNU

Morris et al. (2001) showed that the presence of contingent links in an STNU requires new kinds of constraint propagation when checking dynamic controllability. Those kinds of rules will also be needed for a CSTNU. However, in addition, the presence of observation nodes requires new kinds of propagation rules. One such rule is presented below.

Consider the CSTNU fragment in Fig. 4, where $0 \leq w$, $v \leq w$, α, β and γ are labels that do not share any propositional letters, and p is a propositional letter that does not appear in α, β or γ. The time-point, $p?$, is the observation time-

[9] In the earlier paper, they incorporated contingent links and presented a preliminary extension of their dispatchability algorithm.

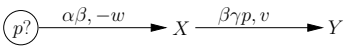

Figure 4: The context for label modification in a CSTNU

point for p. Thus, when $p?$ executes, the value of p becomes known. The arrow from $p?$ to X represents the labeled constraint, $(X - p? \leq -w,\ \alpha\beta)$. Thus, in scenarios where $\alpha\beta$ is true, $X + w \leq p?$ must hold. Thus, in those scenarios, X must be executed before p is observed. The arrow from X to Y represents the constraint, $(Y - X \leq v,\ \beta\gamma p)$. Thus, in scenarios where $\beta\gamma p$ is true, $Y \leq X + v$ must hold—in which case, Y must execute before the value of p is known.

Lemma 8 (Label Modification Rule)**.** *If σ is a dynamic execution strategy that satisfies the labeled constraints in Fig. 4 in all scenarios where their labels are true, then σ must also satisfy the labeled constraint, $(Y - X \leq v,\ \alpha\beta\gamma)$, in all scenarios where $\alpha\beta\gamma$ is true. Moreover, the original labeled constraint, $(Y - X \leq v,\ \beta\gamma p)$, can be replaced by the pair of labeled constraints, $(Y - X \leq v,\ \alpha\beta\gamma)$ and $(Y - X \leq v,\ (\neg\alpha)\beta\gamma p)$.*

Proof. Let σ be as in the statement of the lemma. However, suppose that there is some drama, (s, ω), such that: (1) the label $\alpha\beta\gamma$ is true in scenario s; but (2) the schedule, $\sigma(s, \omega)$, does *not* satisfy the constraint, $(Y - X \leq v)$. Let s_2 be the same scenario as s, except that the value of p is flipped. Now, by construction, in one of the scenarios, s or s_2, the label, $\alpha\beta\gamma p$, is true. Let $\hat{s}$ be that scenario, and $\sigma(\hat{s}, \omega)$ the corresponding schedule. By construction, that schedule satisfies both of the labeled constraints from Fig. 4, since their labels are true in $\hat{s}$. Thus,

$$\begin{aligned}
[\sigma(\hat{s},\omega)]_Y &\leq [\sigma(\hat{s},\omega)]_X + v\ , \quad \text{since } Y - X \leq v \\
&\leq [\sigma(\hat{s},\omega)]_X + w, \quad \text{since } v \leq w \\
&\leq p? \qquad\qquad\quad , \quad \text{since } X - p? \leq -w
\end{aligned}$$

Let $\tilde{s}$ be the scenario that is the same as $\hat{s}$, except that the value of p is flipped. Let t be the first time at which the schedules, $\sigma(\hat{s}, \omega)$ and $\sigma(\tilde{s}, \omega)$, differ. Thus, there must be some time-point T that is executed in one of the schedules at time t, and in the other at some time later than t. But in that case, the corresponding histories at time t must be different. But the only possible difference must involve the value of the proposition p, since all other propositions and contingent durations are identical in the dramas, $(\hat{s}, \omega)$ and $(\tilde{s}, \omega)$. Thus, $p?$ must be executed before time t. Since t is the time of first difference in the schedules, it follows that $p?$ is executed at the same time in each of these schedules. Furthermore, since X and Y are both executed before $p?$ in $\sigma(\hat{s}, \omega)$, and hence before the time of first difference, it follows that X and Y are also executed at those same times in $\sigma(\tilde{s}, \omega)$. Thus, regardless of the value of p, the constraint $Y - X \leq v$ is satisfied, contradicting the choice of (s, ω).

For the second part, consider the following constraints:

- C_1: $(Y - X \leq v,\ \beta\gamma p)$
- C_2: $(Y - X \leq v,\ \alpha\beta\gamma)$
- $C_{1.1}$: $(Y - X \leq v,\ \alpha\beta\gamma p)$
- $C_{1.2}$: $(Y - X \leq v,\ (\neg\alpha)\beta\gamma p)$

C_1 is the constraint from X to Y shown in Fig. 4. C_2 is the constraint derived in the first part of this proof. Now, C_1 is equivalent to the pair of constraints, $C_{1.1}$ and $C_{1.2}$, since $\beta\gamma p \equiv (\alpha\beta\gamma p) \vee ((\neg\alpha)\beta\gamma p)$. Thus, the constraint set $\{C_1, C_2\}$ is equivalent to the constraint set $\{C_{1.1}, C_{1.2}, C_2\}$. However, since the label on C_2 is subsumed by the label on $C_{1.1}$, the constraint C_2 *dominates* the constraint $C_{1.1}$. Thus, the constraint set $\{C_1, C_2\}$ is equivalent to $\{C_{1.2}, C_2\}$. □

This and other label-modification rules are expected to play an important role in the dynamic controllability-checking algorithm that is a major goal of this work.

Conclusions

This paper presented a temporal network, called a CSTNU, that generalizes CSTPs and STNUs from the literature. The semantics of dynamic controllability for CSTNUs also generalizes the related notions for CSTPs and STNUs. The motivation for this work was to provide a framework for the temporal constraints underlying workflows for business and medical-treatment processes. In future work, we aim to show that any workflow is history-dependent controllable if and only if its underlying CSTNU is dynamically controllable.

References

Bettini, C.; Wang, X. S.; and Jajodia, S. 2002. Temporal reasoning in workflow systems. *Distributed and Parallel Databases* 11:269–306.

Combi, C., and Posenato, R. 2009. Controllability in temporal conceptual workflow schemata. In Dayal, U. et al., ed., *Business Process Management*, volume 5701 of *Lecture Notes in Computer Science*. Berlin, Heidelberg: Springer-Verlag. 64–79.

Combi, C., and Posenato, R. 2010. Towards temporal controllabilities for workflow schemata. In *Proc. of the 17th International Symposium on Temporal Representation and Reasoning (TIME-2010)*, 129–136. Los Alamitos, CA, USA: IEEE Computer Society.

Combi, C.; Gozzi, M.; Juárez, J.; Oliboni, B.; and Pozzi, G. 2007. Conceptual modeling of temporal clinical workflows. In *Proceedings of the TIME-2007 workshop*, 70–81. IEEE Computer Society.

Conrad, P. R., and Williams, B. C. 2011. Drake: An efficient executive for temporal plans with choice. *Journal of Artificial Intelligence Research* 42:607–659.

Conrad, P. R. 2010. Flexible execution of plans with choice and uncertainty. Master's thesis, Massachusetts Institute of Technology.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Hunsberger, L. 2009. Fixing the semantics for dynamic controllability and providing a more practical characterization of dynamic execution strategies. In *Proceedings of the TIME-2009 workshop*, 155–162. IEEE Computer Society.

Morris, P.; Muscettola, N.; and Vidal, T. 2001. Dynamic control of plans with temporal uncertainty. In Nebel, B., ed., *17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, 494–499. Morgan Kaufmann.

Morris, P. 2006. A structural characterization of temporal dynamic controllability. In *Principles and Practice of Constraint Programming (CP 2006)*, volume 4204 of *Lecture Notes in Computer Science*. Springer. 375–389.

Tsamardinos, I.; Vidal, T.; and Pollack, M. 2003. CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints* 8(4):365–388.

On-line Simulation based Planning

Luis Gustavo Rocha Vianna **Leliane Nunes de Barros** **Karina Valdivia Delgado**
University of Sao Paulo, Sao Paulo, Brazil.
leliane@ime.usp.br, ludy17ludy@gmail.com, kvd@usp.br

Abstract

Intelligent agents in real environments are required to act quickly and to deal with the uncertainty about the world. Nevertheless, they must act in an effective way to achieve their given goal. A possible solution for this problem is to have an on-line planner that can interleave planning and execution to rapidly discover a near optimal first action without having to determine an optimal policy. In this paper, we discuss what is required from an online planner, present some strategies that are suited for this type planning and select two state-of-the-art algorithms, and their extensions, for solving probabilistic planning problems based on simulation: *Real Time Dynamic Programming* (RTDP) and *Upper Confidence bounds applied to Trees* (UCT)

Introduction

Probabilistic planning problems are difficult to solve, and, usually, there is not enough time to find an optimal plan. In this case, interleaving planning and acting can lead to better results than offline planning. There are four important aspects that an online planning agent must be concerned about. i) Focus its search around the initial state and look for the best initial action visiting as few states as possible. ii) Maximize the expected reward even considering we won't find the optimal action, i. .e, discard very bad choices quickly and keep the probability of chosing a near optimal action high. iii) Reason about the planning and acting times, the should have a ways to know if he has found the optimal action, or how close to it, or how long it will probably take to find it. For instance, have a bound on the value function and an estimate of how many iterations will be needed to converge some states. iv) Think big, even when at any planning step we only need to find the best action, a planning agent should control its resources considering all the problem and try to use every step to make the following steps easier and eventually find the optimal solution. These are interesting but complex goals for a multi-purpose online planning agent and we will see how some of these can be followed.

In this paper, we have selected two state-of-the-art algorithms for solving probabilistic planning based on simulation to discuss the properties they have to efficiently interleave planning and execution. The selected algorithms are: *Real Time Dynamic Programming* (RTDP) and *Upper Confidence bounds applied to Trees* (UCT). Since the winners (1st, Prost (Keller and Eyerich 2012) and 2nd, Glutton (Andrey Kobolov and Weld 2012b), respectively) of the IPPC 2011 (The International Probabilistic Planning Competition of 2011) (Sanner and Yoon 2011) were based on these algorithms, UCT and RTDP respectively. they provide many insights to efficiency improvements on probabilistic planning. . We first define a formal model of the planning task as a *Markov Decision Process* (MDP) and give a brief description of the RTDP (Barto, Bradtke, and Singh 1993) and UCT (Kocsis and Szepesvri 2006) planning algorithms. After that, we present some possible requirements and properties that a planning and execution problem might have and discuss how RTDP and UCT planners can deal with them.

Probabilistic Planning Problem

A very used mathematical model for probabilistic planning problems is the Markov decision process (MDP), which is formally defined as a tuple $\langle S, A, T, R, \gamma \rangle$ (Puterman 1994), where $S = \{s_1, \ldots, s_n\}$ is a finite set of fully observable states; $A = \{a_1, \ldots, a_m\}$ is a finite set of actions, $T : S \times A \times S \to [0, 1]$ is a known stationary Markovian transition probability function; $R : S \times A \to \mathbb{R}$ is a fixed known reward function associated with every state and action; and γ is a discount factor s.t. $0 \leq \gamma \leq 1$ where rewards t time steps in the future are discounted by γ^t. In a probabilistic planning problem, there is a set of initial states $\mathcal{I} \subseteq S$, and a possibly empty set of absorbing goal states $\mathcal{G} \subset S$ where all actions lead to a zero-reward self-transition with probability 1, this special subclass of MDPs is called Shortest Stochastic Path (SSP). For some planning problems, the goal is to accumulate reward before achieving a given goal state, for others, the goal is to maximize the expected discounted reward for a time horizon H. In a finite horizon MDP we define the augmented state space $S_H = S \times \{0, 1, 2..H\}$, of pairs state and number of remaining steps.

A policy $\pi : S \to A$ (or $\pi : S_H \to A$) specifies the action $a = \pi(s)$ to take in state $s \in S$ (or $s \in S_H$). Our goal is to find an optimal policy π^* that maximizes the value function, defined as the sum of expected discounted rewards following that policy.

$$V_\pi(s) = E_\pi \left[\sum_{t=0}^{H} \gamma^t \cdot r_t \Big| s_0 = s \right] \quad (1)$$

with $s \in S$ if $H = \infty$ and $s \in S_H$ otherwise, and where r_t is the reward obtained at time t, and we define $H = \infty$ for infinite horizon problems. We define $V^*(s)$ the optimal value of the state s equals to the value it achieves in the optimal policy, $V^*(s) = V_{\pi^*}(s)$.

Let X be S or S_H for infinite or finite MDPs respectively. The quality($Q_\pi(s,a)$) of an action $a \in A$ at a state $s \in X$ under a policy π is defined as the expected reward of applying action a at s assuming policy π will provide the next rewards.

$$Q_\pi(s,a) = R(s,a) + \gamma * \sum_{s' \in X} P(s,a,s') * V_\pi(s') \quad (2)$$

where R is the reward function and P is the probabilistic transition function. We also define the optimal quality $Q^*(s,a)$ of a at s as the quality achieved by the optimal policy, $Q^*(s,a) = Q_{\pi^*}(s,a)$.

Regret is the difference between the discounted reward obtained by the optimal policy π^* and a given policy π, i.e.:

$$regret(\pi, s) = V^*(s) - V_\pi(s), s \in X \quad (3)$$

The regret of one possible first action a is the minimum regret of a policy π containing that first action. Moreover, it is how much reward is lost by choosing this action instead of the optimal, the difference between the optimal action quality and this action's quality.

$$regret(a,s) = \min_{\pi | \pi(s)=a} regret(\pi, s) = Q^*(s, \pi^*(s)) - Q^*(s,a) \quad (4)$$

Simulation Based algorithms

On-line simulation based algorithms perform a search of the best action to be executed in the current state. They initially consider the immediate rewards and make simulations to obtain information about the possible rewards for each of the available actions and attempt increase the expected rewards, or equivalently, to minimize the regret of the next action to be executed. First we'll give an outline on the basis of the selected algorithms and then move to online planning strategies. Although the RTDP planning algorithm (Barto, Bradtke, and Singh 1993) has been proposed to produce optimal policies, it can be an on-line algorithm, as we explain in the next section. The UCT (Kocsis and Szepesvri 2006) planning algorithm has already been proposed to be an on-line planner, performing a large number of simulations to efficiently estimate the quality of every action on the current state with a search horizon h.

RTDP planning

One of the state-of-the-art solutions for MDP problems is the real-time dynamic programming (RTDP) (Barto, Bradtke, and Singh 1993) algorithm. RTDP uses asynchronous dynamic programming approach that applies the Bellman update to states in an arbitrary order, that is, it updates states encountered during trial-based simulations of an MDP. The simulation trials explore the state space in a sequence of states, updating the value of each visited state. RTDP selects the next state to be visited (Figure 1) by drawing next state samples s' from the transition distribution for the current greedy action a and current state s, i.e.,

$$\text{CHOOSENEXTSTATE}(s,a) := s' \sim T(s,a,\cdot). \quad (5)$$

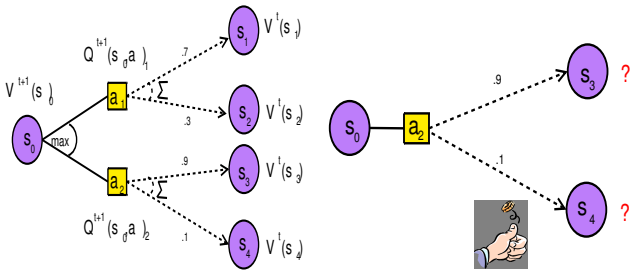

Figure 1: a) RTDP Bellman update and b) RTDP next state choice.

Given an admissible upper bound V_h on the optimal value $V^*(s)$ (i.e., $V_h(s)' \geq V^*(s); \forall s$), RTDP converges to the optimal value function in the infinite limit of trials. However, in a planning and execution setting, the RTDP algorithm can be interrupted at any time, generally yielding a better solution the longer it is allowed to run.

One weakness of RTDP is that unlikely paths tend to be ignored and as a consequence the convergence of RTDP is slow (Bonet and Geffner 2003). Thus, some extensions of RTDP were proposed in order to improve the convergence such as Labeled RTDP (LRTDP) (Bonet and Geffner 2003) and Bounded RTDP (BRTDP) (McMahan, Likhachev, and Gordon 2005).

Upper Confidence bounds applied to Trees (UCT)

The UCT algorithm was proposed by Kocsis and Szepesvari (2006) to solve MDPs. It is an on-line planning algorithm where a large number of simulations are used to estimate the quality ($\bar{Q}^*(s,a,h)$) of the best action on the current state with a search horizon h, i.e., the total discounted reward in h steps for each possibly applied action in the current state. The estimative is based on the sampled rewards obtained in *rollouts* (similar to RTDP trials), episodic simulations, i. e. sequence of state, action, reward triplets according to the MDP model, with depth equal to the search horizon. These *rollouts* are performed following a heuristic or simply random policy to decide for an action and using model simulator calls to obtain a new state. Like the RTDP trials, *Rollouts* occur in a limited search horizon. After finishing a simulation (performing a set of *rollouts* for each decision step), the action with greatest estimated reward is selected.

The UCT algorithm (Kocsis and Szepesvri 2006) builds a tree (Figure 2) by storing previously sampled results to guide the choices in each node. The nodes of the tree correspond to pairs (state, search horizon). Initially the tree contains only the initial state with the desired search horizon as a single leaf. The UCT search strategy uses two policies (Figure 2 b)): (i) a tree policy, that is applied in the explored part of the tree choosing actions according to previous result until reaching a tree leaf and; (ii) the *rollout* policy, applied only to a tree leaf choosing one of the available actions and performing a *rollout* following a random policy to retrieve a total discounted reward. UCT uses this value to estimate the quality of the chosen action. The number of times each action was chosen is used to calculate the average of the obtained rewards to estimate $Q^*(s,a,h)$. After the rollout is finished for a leaf, the reward obtained is back-propagated

upwards in the tree.

$$\bar{Q^*}(s,a,h) = \frac{(occurrences(s,a,h) * \bar{Q^*}(s,a,h) + Rew}{occurrences(s,a,h)+1} \quad (6)$$

where Rew is the total reward obtained from the rollout or the corresponding value back-propagated from a leaf upwards and $occurrences(s,a,h)$ is how many times this action was updated, incremented by one at every update.

The tree policy decides which action to choose based on the upper bound of the actions quality (Auer, Cesa-Bianchi, and Fischer 2002) using the exploration bias modifier constant C in the following equation:

$$TreePolicy(s,h) =$$

$$argmax_{a \in A} \left\{ \bar{Q}(s,a,h) + C\sqrt{\frac{ln(occurrences(s,h))}{occurrences(s,a,h)}} \right\}$$

UCT planning is an approach that can find near-optimal solutions for large state-space planning problems. Thus, using UCT as an on-line algorithm, it can return the most promising next action.

Online Planning

Many planning algorithms allow online planning setting, by choosing a best action for the current state s. Offline planning aims to solve the problem completely, obtaining a closed optimal policy that guarantees that no further planning will be needed and executing the policy will lead the best results. When executed online, a planning algorithm does not need to compute a complete or even closed policy, it only needs to focus on the choice of the action to execute in the current state. After executing the chosen action it will plan again for the next state, possibly using information of the previous computations to make the choices better and faster, so that it uses less resources, in the following steps. Executing an action can lead the agent to a smaller state space. Note that in many domains the effects of a single action do not last too long, due to probabilistic effects, for instance on the SysAdmin domain from IPPC2011 even if we have recently rebooted a computer it may soon stop running, making long plans for this domain is meaningless because it is hard to predict the actual future states.

Interleaving Planning and Execution

Depending on the planning and execution problem the agent has to solve, there are different requirements that it has to meet, related to time, horizon and quality of its decisions (i.e., with bounded regret), in order to accomplish a given task. In this section we discuss some of this requirements.

Focused Search

In a planning and execution problem, the time an agent can use to decide the next action is usually short and insufficient to complete a total plan, requiring it to restrict its search. Recent strategies for that are:

Initial State Restriction. Online planners should perform only partial updates, i.e. only states reachable from the initial state s_0 should be updated.

Limited Horizon. For several problems with large or infinite horizon it is prohibitive to make unrestricted simulations. Thus considering a smaller horizon to choose the next action can be more efficient. For instance, when solving an infinite or very large horizon MDP, UCT (Kocsis and Szepesvri 2006) or RTDP can chose a smaller horizon size for their simulations. This is good because the simulations are faster and for most problems the current state neighborhood can be more thoroughly explored. However, the optimal policy for a shorter horizon may not be suited for a some long horizon problem, causing possibly great regrets. One such planning domain is the Navigation from IPPC2011(Sanner and Yoon 2011) where there is only non-negative reward for a single goal state and initial choices may lead to a dead-end state. Finding an appropriate horizon is hard and it must be is empirically determined: for dense transition models only small horizons are tractable whereas for a very sparse transition model longer horizons may be more informative. Glutton (Andrey Kobolov and Weld 2012b) overcomes this difficulty by using reverse iterative deepening to solve finite horizon MDPs with increasing horizon. The results from smaller horizons, value function and labeled states, are reused for the next iteration so it solves problems as large as the limited time permits. Another strategy is used by the Short Sighted Probabilistic Planner() is defining and solving smaller t-short-sighted SSP, SSPs with only the states reachable in t steps. Solving a t-short SSP gives the greedy policy for the next t steps and allows for simulation of the next t steps. This can reduce the number of planning stages on a simulation and find a good approximation for the value function for states in a t neighborhood of s_0.

Bounded Error

Another possible assumption about planning time would be a scenario where the agent is supposed to return only a safe action in the least time, that is, the agent can only tolerate a predefined ϵ maximum regret (Equation 4), however it must act as soon as possible. As UCT does not have guarantees on expected error, it can only know that increasing the number of rollouts the chance of choosing a suboptimal action decreases. Thus, UCT is not fit for bounded error applications.

Bounded RTDP (BRTDP) (McMahan, Likhachev, and Gordon 2005) maintains upper and lower bounds on the optimal value function, $V_u(s)$ and $V_l(s)$ respectively, and focus search in areas with high value uncertainty. The gap between upper and lower bounds provides a measure of the value uncertainty for state s. BRTDP finishes a trial if finds a goal state or at a limited depth or there is low value uncertainty for all states. An interesting property of BRTDP for a planning and execution setting is that the uncertainty on the state value can also be used to know how close we are to the optimal value of the current state and then choose an action with some optimality guarantee: a small gap between upper and lower bound means we are near to the optimal action choice.

Efficient time Managing

An online planning agent must be able to do its best in a given time.

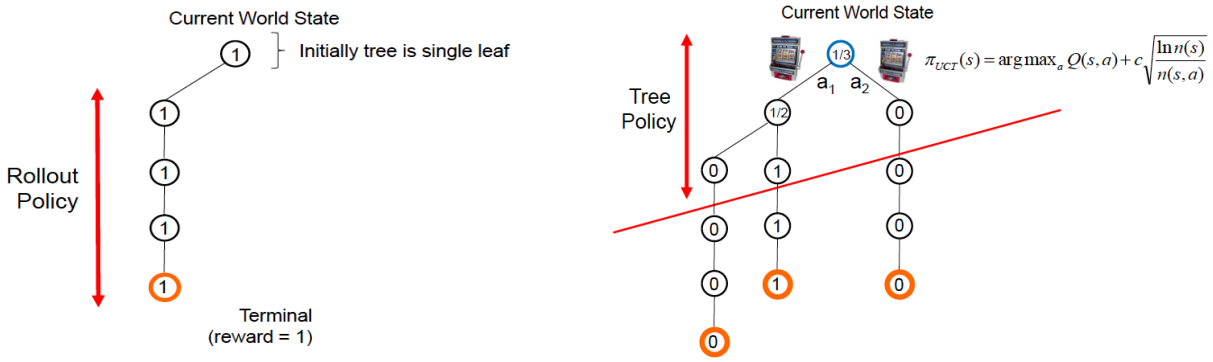

Figure 2: a) UCT rollout and b) UCT tree policy (Extracted from *http://videolectures.net/icaps2010_fern_mcpbprp/*).

Fixed time. In a real world problem we can also assume we have a fixed time t before acting. For instance, in the Mars Rover domain, the Rover must take n pictures during the light time. When the agent starts to plan, it knows how much time it has and this information can be used to make an efficient plan. For instance, an online planner in a finite horizon MDP knows how many steps he will need to perform and it can distribute the total time among the steps considering that the final steps will need less time due to their smaller remaining horizon and the accumulated information. A practical example is Gourmand (Andrey Kobolov and Weld 2012a) whose iterative deepening strategy provides estimates on how long it takes to solve a finite horizon MDP for each horizon h. Gourmand initially gives equal time to all steps, but as more information is gathered it considers how much time he can take from each future step without reducing the largest solvable horizon for that step. This spare time can be invested in trying to solve the current step with a greater horizon.

Since UCT and RTDP are any-time algorithms, they both can also work well with a fixed time t returning the near optimal action they can find during this time. However, typically they do not use the time information to improve their performance.

Meta Planning

Solving a planning task online involves a sequence of stages where the agent can plan to decide an action and execute it. The agent must reason to solve each stage, but in order to perform better it should also reason on the global task. An already mentioned strategy is the time distribution among the stages, possibly using estimates of the time needed in each stage, e.g. a function of the remaining horizon (Andrey Kobolov and Weld 2012a). Another common global strategy is the reuse of previous computations. In simulation based algorithms for infinite horizon MDPs the estimates of the value function or action quality can directly be applied for the next stage. However, keeping all this information might consume too much space therefore it is possible to make approximations to compress the reused information. The default policies investigated by GLUTTON(Andrey Kobolov and Weld 2012b) can also be seen as a meta planning strategy. They are simple policies, such as cyclic or random, which can be used as heuristics or be compared to the planner's solution, eventually chosen instead of the calculated policy.

Conclusion

The main motivation of this paper is to discuss the desired features for online probabilistic planning. Specifically, online planners need to focus their search on the relevant states w.r.t. s_0 restricted to time, space and bounded error limitations. We show some possibilities to efficiently deal with time constraints and also some meta planning strategies for on-line planning. To illustrate this discussion, we used two very popular approaches, Monte Carlo planning and Real Time Dynamic Programming, and their extensions which implement most of the presented ideas. Since there has not been much literature on using UCT and RTDP in a online setting, this paper tries to put together most of the recent improvements used in the IPPC 2011, plus some insights from our own experience on the area, that we hope to be useful for new developments.

Acknowledgements

This work has been supported by FAPESP grant 2011/16962-0.

References

Andrey Kobolov, M., and Weld, D. 2012a. LRTDP vs. UCT for Online Probabilistic Planning. In *AAAI'12*.

Andrey Kobolov, M., and Weld, D. 2012b. Reverse iterative deepening for finite-horizon mdps with large branching factors. In *ICAPS'12*.

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.* 47(2-3):235–256.

Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1993. Learning to act using real-time dynamic programming. Technical Report UM-CS-1993-002, U. Mass. Amherst.

Bonet, B., and Geffner, H. 2003. Labeled RTDP: Improving the convergence of real-time dynamic programming. In Giunchiglia, E.; Muscettola, N.; and Nau, D., eds., *ICAPS-03*, 12–21.

Keller, T., and Eyerich, P. 2012. PROST: Probabilistic Planning Based on UCT. In *In Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*, To appear.

Kocsis, L., and Szepesvri, C. 2006. Bandit based monte-carlo planning. In *ECML-06*, 282–293. Springer.

McMahan, H. B.; Likhachev, M.; and Gordon, G. J. 2005. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *ICML-05*, 569–576.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York: Wiley.

Sanner, S., and Yoon, S. 2011. International Probabilistic Planning Competition (IPPC). `http://users.cecs.anu.edu.au/ssanner/IPPC_2011/`.

Extended Spectrum Based Plan Diagnosis for Plan-Repair

Shekhar Gupta , Bob Price and Johan de Kleer
Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto CA 94304 USA
Email: {shekhar.gupta,bprice,dekleer}@parc.com

Nico Roos
Masstricht University
The Netherlands
Email: roos@maastrichtuniversity.nl

Cees Witteveen
Department of Software Technology
Delft University of Technology, The Netherlands
Email: c.witteveen@tudelft.nl

Abstract

In dynamic environments unexpected malfunctions or conditions can cause plan failure. Research has shown that plan-repair on failure can be more efficient than building complete conditional plans from scratch to handle all contingencies. The effectiveness of replanning depends on knowledge of exactly which plan actions failed and why. Conventional Model Based Diagnosis (MBD) can be used to detect such faulty components but the modeling cost (to generate the fault model) outweighs the benefits of MBD. In this paper, we propose an Extended Spectrum Based Diagnosis approach that efficiently pinpoints failed actions and does not require the fault models. Our approach first computes the likelihood of an action being faulty and subsequently proposes optimal probe locations to refine the diagnosis. We also exploit knowledge of plan steps that are instances of the same plan operator to optimize the selection of the most informative diagnostic probes. This reduces costs and improves the accuracy of diagnoses.

Introduction

Classical planning assumes that the world is deterministic so that every action produces the intended effects. However, this assumption is not true in real world planning domains where actions can fail because of unexpected events. Execution of a plan will lead to an unexpected goal state if one or more actions are behaving abnormally. When such incidents happen one possible way to achieve the desired goal state is repairing the original plan by adding/removing some actions (van der Krogt and de Weerdt, 2005). For example, consider a multimodal freight logistics system, where a planner such as TIMIPLAN generates optimal plans to deliver goods from one location to another (Flórez et al., 2011). TIMIPLAN has a plan monitoring component checking whether the execution of plans deviates from the expected outcome and triggers a replanning module if needed. To avoid replanning from scratch, the planner uses plan repair to change the initial plan as little as possible. For example, a damaged truck is replaced by a new truck and TIMIPLANS greedily selects such a truck with the least estimated total cost. This, however, assumes full observability of the health state of the trucks which in general might be too costly or in some cases infeasible.

Model based diagnosis is used to infer the set of faulty component(s) in a system from observations and background knowledge (Reiter, 1987). It exploits a descriptive behavioral model of components together with a structural model of how the components are connected to compute the implications of observations. The idea of MBD can be further extended to diagnose faulty actions in a plan where the plan can be seen as a system and the action can understood as a component MINI-MAX (Roos and Witteveen, 2009). This view enables the application of well known diagnosis techniques to plan descriptions. For instance, knowing that a road must be clear for a truck to pass, and observing that a truck has arrived from a distant city allows the system to infer that the road from that distant city is clear even though this cannot be directly observed. Methodology such as the pervasive diagnosis framework (Kuhn et al., 2010), has demonstrated how diagnosis can be performed on systems controlled by plans, but makes the simplifying assumption that the planning goal is a single output for the system and that any failed action has a direct observable effect on the output. It is therefore unsuitable for domains such as the logistics domain where many goals must be achieved simultaneously and action failures have local effects that are only indirectly related to the goals.

While powerful, model-based techniques require accurate fault models which are expensive to develop and in some cases the required data cannot be obtained at all. For example, it may be difficult to model all the ways in which a truck can fail to deliver a package to a destination. The proposed Spectrum Based Diagnosis (SBD) approach makes use of abstract frequency statistics to reveal possible causes of a problem without a fault model of the system. SBD has been successfully applied for software fault localization (Abreu et al., 2009) and hardware diagnosis (Arjan Van Gemund and Abreu, 2011). In our approach we use SBD to determine the health state of a plan step which infers the health state of corresponding action. In the planning domain, it is common for a single plan operator to be instantiated many times for different plan steps. For instance, a transport operation might be instantiated with the same truck to carry packages on several different routes in a plan. All plan steps that are instantiated from the same operator will fail if there is something wrong with the plan operator. For instance, every attempt to schedule a shipment on a blocked road will fail. In the online

replanning context, we are given the plan ahead of time, so we can exploit knowledge about the operator dependencies of actions within a plan. We propose Extended Spectrum Based Diagnosis which is able to exploit available information about such dependencies in the plan by elegantly extending the spectrum matrix. Finally, in domains, such as the logistics domain, we often have the ability to take information gathering actions. Perhaps we could get the dispatcher to call drivers and ask for a report on road conditions along a particular segment. However, each of these actions has costs involved. We address this, by combining our extended spectrum based diagnosis with an optimal probing strategy, which uses a mutual information criteria. Given the resulting information, standard replanting techniques are used to repair the plan. The result is a practical approach to planning for online systems with dynamic failures that works with incompletely described systems but exploits the known information to efficiently repair plans with the lowest cost. In the following sections we develop the mathematical framework for extended spectrum based diagnosis and demonstrate it on a notional multimodal transportation problem.

Preliminaries

Our planning formalism is modeled after the STRIPS planner (Fikes and Nilsson, 1971). Our specific notation is covered in following subsections.

State The world can be described by a finite set $Var = \{v_1, v_2, \ldots, v_n\}$ of variables and their respective *value domains* D_i. A particular state is denoted by an n-tuple $\sigma = (\sigma(v_1), \ldots, \sigma(v_n)) \in D_1 \times D_2 \times \cdots \times D_n$. In multimodal transportation system, the variables would represent the locations of individual items such as trucks and goods to be shipped.

Actions, plan operators and plan steps An *action* refers to an activity that results in some change of the (current) state of the world. A *plan operator* refers to a description of such an action in a plan. More exactly, a plan operator o is a function mapping state (σ_0) to another state (σ_1).

An instantiation of an operator o with specific arguments is called a plan step. It maps a specific state into another specific state. Therefore, given a set O of plan operators, we consider a set $S = inst(O)$ of instances of plan operators in O, called the set of plan steps. A plan step will be denoted by a small roman letter s_i. For example, a plan operator can be understood as a shipping action by a specific mode of transportation, i.e., a truck, a train or a ship. Such a shipping action can be used at several places in the plan using the same truck. Each specific occurrence of such a truck transportation is a plan step.

If plan step s is an instantiation of operator o, we say that $o(s) = o$. If for two plan steps s and s' it holds that $o(s) = o(s')$ they are said to be *related* to each other. In other words, s and s' are sharing same resource therefore there resource dependency between these two plan steps. For example, if the same truck (plan operator) is used to execute two different transportations (plan steps), these plan steps are related. Note that here plans differ from systems where normally components operate quite independently from each other. In plans, it seems rational to assume that a structural fault in the truck might affect at least a subset of its instantiations (plan steps).

If two plan steps are instantiated from the same operator, we say that they are related. Let $o(s)$ be the operator that step s is instantiated from. Given two plan steps s and s', if $o(s) = o(s')$ then they are related. For example, if the same truck (plan operator) is used to execute two different transportations (plan steps), these plan steps are related. Note that unlike typical physical systems in which components that make up the system fail independently, plans which contain related plan steps need to model the dependence in the failures between the related plan steps.

Plan and plan execution We represent our plans as a partially ordered set of steps. Formally, a plan is a tuple $P = \langle \mathcal{O}, S, < \rangle$ where $S \subseteq inst(\mathcal{O})$ is a set of plan steps occurring in $\mathcal{O}$ and $(S, <)$ is a partial order (Cox, Durfee, and Bartold, 2005). If step $s' < s$ then s' must be executed before s. Same $<$ relation can be used to denote the relative order between states. Figure 1 gives an illustration of a partially ordered plan.

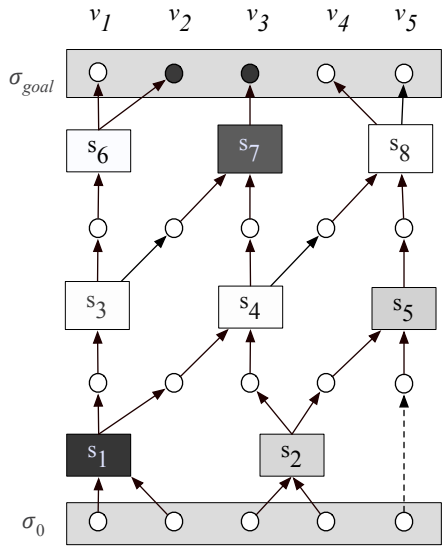

Figure 1: A partially ordered plan graph in which an initial state σ_0 is transformed by plan steps (s_i for $i = 1, 2, \ldots, 8$) into a goal state σ_{goal}. Each state characterizes the values of five variables v_1 , v_2 , v_3 , v_4 and v_5 . Plan steps having the same color (e.g. s_1 and s_7, and s_2 and s_5 are instantiations of the same plan operator.

Observations Our framework enables us to observe a set of values of the variables making up a state of the world. We denote an observation of variable v in state σ by σ_v. We assume that a cost is associated with any observation, except for observing the initial state (σ_0) and the goal state σ_{goal}. For ease of exposition, we assume all probes have equal cost.

Plan Diagnosis

We use conventional MBD notation to represent the plan we are diagnosing.

Definition 1 *A system is a pair (P, OBS) where P is a plan tuple $\langle \mathcal{O}, S, < \rangle$ and OBS is the set of values of the observed variables at the initial state σ_0 and the goal state σ_{goal}.*

Plan execution is validated by continuously monitoring the goal state. The difference in the observed value $\sigma_{goal(v)'}$ of any variable v in the goal state from the expected value $\sigma_{goal(v)}$ implies the plan execution failure, i.e., some plan steps are not executed in a correct way. For example, consider the plan shown in Figure 1. Suppose this plan represents a multimodal transportation plan where five goods $(v_1 \dots v_5)$ need to be delivered from initial location to goal location using different transportation modes $(s_1 \dots s_8)$. In the final destination, it is observed that two goods (v_2 and v_3) have not arrived which implies one or more plan steps are faulty.

Let $h_j \in \{ok, ab\}$ be the health state of plan step s_j where ok represents normal behavior and ab abnormal behavior. In establishing which part of the plan fails, we are only interested in those plan steps qualified as abnormal. Therefore, a plan diagnosis can be defined as following:

In establishing which part of the plan fails, we are only interested in those plan steps qualified as abnormal. Therefore, a plan diagnosis can be defined as following:

Definition 2 (Diagnosis) *A diagnosis P_D of a plan $P = \langle \mathcal{O}, S, < \rangle$ is a tuple $P_D = \langle \mathcal{O}, S, <, D \rangle$, where $D \subseteq \mathcal{S}$ is the subset of plan steps qualified as abnormal (and therefore, $S - D$ is the subset of plan steps qualified as ok).*

Spectrum Based Diagnosis

In absence of a detailed fault model of plan operators and plan steps, SBD is a suitable diagnosis methodology for the problem in hand. The basic principle of SBD can be described as follows: if the value of a variable in the goal state is incorrect, then one or more plan steps involved in generation of that variable are abnormal.

Obtaining the Spectrum Matrix The spectrum matrix shows for every variable in σ_{goal} which plan steps are involved from the state σ_0 to σ_{goal}. It records, in the goal state, whether a particular variable v_i has the expected value or not. Together with the information about involvement of plan steps, the resulting spectrum gives debuggers hints about the plan steps which are more likely related to failure, and hence have higher possibility to contain the faults.

The spectrum matrix (A, e), where $A = [a_{ij}]$ is the plan spectrum and e is the error vector can be constructed as follows: The plan spectrum A has N rows (one for each variable) and M columns (one for each plan step). We have $a_{ij} = 1$ if a plan step s_j is involved in the generation of variable v_i in σ_{goal}, else $a_{ij} = 0$. The vector e stores whether the outcome for variable v_i has the expected value $(e_i = +)$ or not $(e_i = -)$.

For example, suppose that in the plan presented in Figure 1, the value of variable v_2 and v_3 is not what we would expect in the goal state. Therefore $e_i = -$ for $i = 2$ and $i = 3$ and the following spectrum matrix can be obtained:

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	e
v_1	1	0	1	0	0	1	0	0	+
v_2	1	0	1	0	0	1	0	0	-
v_3	1	1	1	1	0	0	1	0	-
v_4	1	1	0	1	1	0	0	1	+
v_5	1	1	0	1	1	0	0	1	+

In any row with an unexpected outcome, at least on of the components used must be faulty. A minimal hitting set algorithm, STACCATO (Abreu et al., 2009), can be applied to the set of rows with unexpected outcomes to generate the set of diagnoses candidates (c_k) $\{c_1 =< s_1 >, c_2 =< s_3 >, c_3 =< s_2, s_6 >, c_4 =< s_4, s_6 >, c_5 =< s_7, s_6 >\}$.

The Spectrum Matrix for Plan Steps with Shared Resources The candidate $< s_2, s_6 >$ implies that the operator associated with s_2 may be faulty but it could be expensive or difficult to probe the output of s_2. From our knowledge of the plan, we know that s_2 is instantiated from the same operator as s_5. Therefore s_5 is also likely to fail, if s_2 fails. In this case, the failure of s_5 may have been intermittent or the failure may not have been relevant to the preconditions of the subsequent step s_8 so it did not have an effect on the final goal state σ_{goal}. This is called a masked fault and it is not picked up by standard SBD methods. This insight is important, because probing at s_5 may be easier and cheaper than probing at s_2. Imagine a scenario in which steps s_2 and s_5 use the same truck. Suppose in s_2, the truck is used at a distant location where it is difficult to inspect. If it is later used in a plan step s_5 at a location with inspection facilities it will be much easier to measure the health of this resource. There is one small complication. If an operator is used more than once in a plan, it could be heathy earlier in the plan and then fail at some later point.

To take these related plan steps into account, we modify the spectrum matrix in such a way that these relations are encoded in the matrix A itself. Suppose that the plan steps s and s' are related. If s is detected as faulty and $s < s'$, it seems reasonable to consider s' as faulty as well. Formally, we calculate the extended spectrum matrix $A' = [a'_{ij}]$ from A as follows:

$$a'_{ij} = \bigvee_{j' < j, o(j') = o(j)} a'_{ij'} \vee a_{ij} \quad (1)$$

In the plan depicted in Figure 1, plan steps with the same background are related. So s_1 and s_7 are related and s_2 and s_5 are related. The extended spectrum matrix would be (new entries appear in bold face):

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	e
v_1	1	0	1	0	0	1	**1**	0	+
v_2	1	0	1	0	0	1	**1**	0	-
v_3	1	1	1	1	**1**	0	1	0	-
v_4	1	1	0	1	1	0	**1**	1	+
v_5	1	1	0	1	1	0	**1**	1	+

Similar to other MBD engine our diagnosis engine assumes that plan steps are failing independently while computing posterior probability for every diagnosis. Therefore,

if we have a diagnosis in plan steps are related to each other our engine will compute incorrect posteriors. Hence diagnosis must not contain related plan steps. The extended matrix ensures that application of MHS algorithm on that matrix will produce diagnosis comprises of independent plan steps.

Application of minimal hitting set algorithm on extended matrix A' will generate diagnoses candidates (c_k) $\{c_1 =< s_1 >, c_2 =< s_3 >, c_3 =< s_7 >, c_4 =< s_2, s_6 >, c_4 =< s_4, s_6 >, c_5 =< s_7, s_6 >, c_6 =< s_5, s_6 >\}$.

Theorem 1 *Introducing related plan steps into the extended matrix ensures that the MHS algorithm will never return a diagnosis that includes two related plan steps.*

Proof Two plan steps will only appear together in a diagnosis if they individually explain distinct error observations. When we insert a pseudo observation for one of the steps into the matrix, the second step becomes an explanation for both error outputs and becomes a singleton diagnosis breaking up the joint diagnosis. Schematically,

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

Keeping related actions from appearing in the same diagnosis prevents us from multiplying these correlated failures together as if they were independent failures. This preserves the accuracy of the diagnosis. The diagnosis set for the extended matrix is $c_k = \{c_1 =< s_1 >, c_2 =< s_3 >, c_3 =< s_7 >, c_4 =< s_2, s_6 >, c_4 =< s_4, s_6 >, c_5 =< s_7, s_6 >, c_6 =< s_5, s_6 >\}$.

Probability Calculation Having corrected the spectrum matrix, we can use the BARINEL (Abreu et al., 2009) diagnostic engine to compute a fault probability for every diagnosis candidate using Bayes rule. For each variable observation $\sigma_{goal(v_i)}$, the posteriors are update according to the following rule for every candidate c.

$$Pr(c_k|\sigma_{goal}(v_i)) = \frac{Pr(\sigma_{goal}(v_i)|c_k) \cdot Pr(c_k|\sigma_{goal}(v_{i-1}))}{Pr(\sigma_{goal}(v_i))} \quad (2)$$

.

The recursion bottoms out with the prior for the candidate, $Pr(c_k)$, which is computed from the individual step priors assuming independent failures. Note that the candidate $< s_i >$ implies health variable $h_i = ab$. Generally:

$$Pr(c_k) = \prod_i \begin{cases} p_i & \text{if } h_i = 1 \\ 1 - p_i & \text{otherwise} \end{cases} \quad (3)$$

where p_i is the prior probability that plan step s_i is faulty.[1]

The BARINEL engine propagates failure probabilities along the plan step dependencies to calculate the probability $Pr(\sigma_1(v_i)|c_k)$ for each output variable i using maximum likelihood estimation (Abreu, 2009). The final posterior probability is computed by combining Equations 2, 3 and $Pr(\sigma_1(v_i)|c_k)$, and fault probabilities are assigned to plan step as shown in Table 1.

[1]In our case, the prior probability of every plan step is assumed to be to be 0.1.

s_i	$Pr(s_i)$	$Pr'(s_i)$	$I(X;Y)$	$I'(X;Y)$
s_1	0.200	0.160	0.512884	0.512884
s_2	0.002	0.002	0.008762	0.008762
s_3	0.800	0.762	0.016707	0.016707
s_4	0.002	0.002	0.264348	0.264348
s_5	0.000	0.002	**0.004198**	**0.011041**
s_6	0.007	0.008	0.000000	0.000000
s_7	**0.003**	**0.160**	0.000000	0.000000
s_8	0.000	0.000	0.074128	0.074128

Table 1: $Pr(s_i)$ and $I(X;Y)$ are derived for original matrix A. $Pr'(s_i)$ and $I'(X;Y)$ are derived for extended matrix A'

Probing Strategy

A major challenge for a diagnostician is to identify a suitable location for a new probe. In conventional MBD, mutual information criterion can be used to evaluate and compare measurement choice based on their information contribution (de Kleer and Williams, 1987), we have adapted this criterion to probing plan based systems with related steps. To illustrate the formulation, assume X is a diagnostic state of a plan and Y is the measure value of a variable at a probing location where X and Y are both random variables. The mutual information between X and Y is defined as:

$$I(X;Y) = \sum_{x,y} \left[p(x,y) \cdot \log \frac{p(x,y)}{p(x)p(y)} \right] \quad (4)$$

For example, suppose we derive mutual information about the value of location $l1$ and $l2$ as $I(X;Y_{l1})$ and $I(X;Y_{l2})$, respectively. In choosing between $l1$ and $l2$, we will choose $l1$ to probe if $I(X;Y_{l1}) > I(X;Y_{l2})$. As described in (Juan Liu and Zhou, 2008), the above expression can be estimated using entropy calculation, which is given as $I(X;Y) = H(Y) - H(Y|X)$, where $H(Y) = \sum_y \left[p(x) \cdot \log \frac{1}{p(x)} \right]$ is the entropy of Y and $H(Y|X) = \sum_{x,y} \left[p(y|x) \cdot \log \frac{1}{p(y|x)} \right]$ is the conditional entropy. For the plan example shown in Figure 1, observations are already given and fault probability has been computed from SBD, shown in Table 1. Estimated fault probabilities and observations in the goal state are used to compute $H(Y)$ and $H(Y|X)$ as described in (Juan Liu and Zhou, 2008). Mutual information for different probing location in our example (Figure 1) is summarized in Table 1.

Exploiting Related Plan Steps in Diagnosis

In the plan described in Figure 1, s_3 has the strongest participation in the unexpected goal state outcomes for variables, v_2 and v_3. In the first column of Table 1, $Pr(s_i)$, we see that the diagnoser assigns s_3 the highest probability of failure. The standard spectrum A assigns different probabilities to plan steps s_1 and s_7. The extended spectrum, which recognizes that s_1 and s_7 are related, increased the fault probability of s_7 and now s_7 and s_1 have equal probability. Similar conclusions can be made for other related plan steps s_2 and s_5.

The mutual information results shown in Table 1 provides us some interesting conclusions. Without any ambiguity both the spectrum matrices suggest that s_1 is the most informative location to probe and that s_7 is the least. Therefore, probing at the output of s_1 is going to improve the diagnosis by the maximum amount. Since s_7 is in the goal state (no cost) of the plan therefore no extra information can be gained which matches our mutual information computation. At the same time, extending the matrix reveals the information content at the output of plan step s_5 to the diagnoser. In this case, s_5 is closer to the middle of the plan than s_2 which means that it better splits the hypothesis space about possible causes of failure and therefore is more informative. In some cases, s_5 may not be more informative, but may be cheaper or easier to measure. In any case, the extended spectrum matrix opens up new options to increase the accuracy and decrease the cost of diagnosis in plans with related plan steps.

Conclusion

Continuous planning in online dynamic real world environments requires accurate diagnosis to pinpoint which plan steps need to be repaired. Spectrum based diagnosis approaches are a natural approach as they do not require explicit fault models to provide useful diagnostic information. We have seen that extended spectrum based diagnosis extends the advantage of traditional spectrum based diagnosis to systems controlled by a plan which can have related plan steps. The extended spectrum matrix also increases the options for probing potentially leading to more accurate and cheaper diagnosis. The technique can be easily extended in many ways such as computing explicit expected probe costs and considering other ways in which operators can be related. Extended spectrum based diagnosis therefore represents an important technology option for robust, practical and efficient plan based control of real world systems.

References

Abreu, R.; Zoeteweij, P.; Golsteijn, R.; and van Gemund, A. 2009. A practical evaluation of spectrum-based fault localization. *Journal of Systems and Software*.

Arjan Van Gemund, S. G., and Abreu, R. 2011. The antares approach to automatic system diagnosis. In *Proceedings of the 22nd International Workshop on Principles of Diagnosis (DX-2011)*, 5–12.

Cox, J. S.; Durfee, E. H.; and Bartold, T. 2005. A distributed framework for solving the multiagent plan coordination problem. In *In AAMAS*, 821–827. ACM Press.

de Kleer, J., and Williams, B. C. 1987. Diagnosing multiple faults. *Artif. Intell.* 32(1):97–130.

Fikes, R., and Nilsson, N. J. 1971. Strips: A new approach to the application of theorem proving to problem solving. In *IJCAI*, 608–620.

Flórez, J. E.; de Reyna, Á. T. A.; García, J.; López, C. L.; Olaya, A. G.; and Borrajo, D. 2011. Planning multi-modal transportation problems. In *ICAPS*.

Juan Liu, Johan de Kleer, L. K., and Zhou, R. 2008. A unified information criterion for evaluating probe and test selection. In *PHM-2008*.

Kuhn, L.; Price, B.; Do, M. B.; Liu, J.; Zhou, R.; Schmidt, T.; and de Kleer, J. 2010. Pervasive diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics, Part A* 40(5):932–944.

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

Roos, N., and Witteveen, C. 2009. Models and methods for plan diagnosis. *Journal of Autonomous Agents and Multi-Agent Systems* 19(1):30–52.

van der Krogt, R., and de Weerdt, M. 2005. Plan repair as an extension of planning. In *ICAPS*, 161–170.

Knowledge Base for Planning, Execution and Plan Repair

Thibault Gateau, Gaëtan Severac, Charles Lesire, Magali Barbier, Eric Bensana
Onera - The French Aerospace Lab - F-31055, Toulouse, France
firstname.lastname@onera.fr

Abstract

Future robotic missions will involve numerous and heterogeneous autonomous robots. The management of these robots, in order to achieve a mission, requires interoperability, not only at low-level communication protocols and operational HMIs, but also at the decision level, to ensure a robust plan execution and repair framework. Through a formal representation of multi-robot knowledge using an ontology, we show how such a model is of high interest in autonomous mission monitoring, as it provides relevant information all along the mission execution and how it helps to define communication and execution protocols from initial planning computation to plan repair in a multi-robot context.

Introduction

Planning, executing and repairing a mission plan is quite easy... in a simulated, deterministic, well-defined and closed environment. In real world, things are more challenging when different autonomous vehicles must achieve together a team mission.

Robots have already reached a high level of autonomy. They are used in many different situations, and most of the time, their autonomy settles on a well-defined embedded software architecture. Many architectures have been proposed in the literature (McGann et al. 2008; Doherty, Kvarnström, and Heintz 2009; Teichteil-Königsbuch, Lesire, and Infantes 2011) and are currently in use for complex real-world situations where autonomous robots must manage the execution of their own tasks.

The advance of the work in this field has led researchers wonder for many years to what extent robots would be able to be integrated into a team, not only concerning one or more human operators, but most importantly other autonomous and heterogeneous robots, with complementary functionalities. Such situations will indeed become more and more common, as robotic missions are becoming more and more complex, looking for more and more autonomy. We can already suppose that robots will be deployed incrementally, because of the cost and time required for setting up such technologies. For instance, in spatial exploration, robots (rovers, satellites) are deployed one after the other, and missions can last a couple of decades. It is reasonable to consider that the number of robot collaborations will increase in the future, for the benefits of science (Visentin 2007).

Robots will then be intrinsically heterogeneous, as they will be developed gradually, and probably by different builders. Hence, their embedded architecture may even be different. Therefore, it would be worthwhile to reuse all the robots' skills, often developed in a mono-robot concern, to allow a team of robots to achieve a specific mission, without re-designing all their control architecture. But, even assuming that this interoperability issue is solved, we are then confronted with the existing gap between planning and executing a real mission in nearly complete autonomy. Having a team of autonomous vehicles achieve a mission needs first the computation of a mission plan. Then, in order to execute this plan and monitor the execution, a *protocol* must be defined to interface "plan actions" to "robot's tasks". Finally, while executing such a plan, failures *will* occur, as the autonomous vehicles will have to face environment disturbances whatever the assumptions made in the plan are. So, plan-repair (or complete re-plan) is compulsory in achieving mission goals. In order to generate *on-line* a relevant planning model, information about the current state must be gathered from the vehicles. Again, interfaces are required between the individual heterogeneous architectures and the planning system.

In this paper, we propose a formalization of knowledge representation in multi-robot autonomous missions. This formalization aims to lay the basis for interface needs in planning, plan execution and plan repair for a team of autonomous vehicles.

State of the art

Mission execution involving several autonomous vehicles has been widely studied. We examine in the following state of the art how links between planning and execution are considered, and how on-line replanning activities are managed.

The BErkeley AeRobot (BEAR) project[1] studies multi-agent probabilistic pursuit-evasion games with heterogeneous robots. In this project, (Vidal et al. 2002) developed a distributed, hybrid and hierarchical architecture system,

[1] http://robotics.eecs.berkeley.edu/bear

in order to manage partial knowledge of states among the team. They take into account a dynamic environment, heterogeneous agents, faults on robots, and sensor imperfections. Contrary to the *sense-model-plan-act* architectures, robot's dynamics is taken into account at the higher level of the decision process, and they take a particular care to limit communications to the minimal needs. However, in spite of the modular aspect of this architecture, connections between modules are numerous, which leads to a complex adaptation process to re-use existing mono-agent architectures. The environment knowledge is well defined for feeding a tactical and a strategy planner, but specific to the current mission.

The ALLIANCE architecture (Parker 1998) is a behavior-based approach to cooperation and allows the robot team members *to respond robustly, reliably, flexibly and coherently to unexpected environmental changes*. They demonstrate successfully the feasibility of their architecture in an implementation example. The robots are of the same type (but with different abilities) and are controlled by the same architecture. In the same way, (Chaimowicz et al. 2001) proposes an architecture system for tightly coupled multi-robot cooperation. The robot team is organized in a flexible leader-follower structure in which the local robot architecture is independent of the robot control manager. The system is designed for a specific mission.

In Hierarchical Task Network-based approaches (Nau et al. 2003), the planning problem is modeled in a hierarchical way. The Retsina architecture (Paolucci, Shehory, and Sycara 2000) manages the execution of such a hierarchical plan: a dynamic list of objectives is stored as a priority queue, and each agent selects a task to be executed. Thanks to their ability to achieve planning tasks interleaved with executive tasks, the agents are thus highly adaptable to the environment. Concrete interaction with a real environment is not yet described, and they seem to remain only in a planning point of view. Partial replanning is addressed by inserting new tasks into the priority queue in order to have a new resolve of the partial planning problem. Nevertheless, they do not manipulate an explicit global team plan, making it difficult to reason at team level and to manage team organization. (Fazil Ayan et al. 2007) emphasizes local repair in such a hierarchical plan structure in a mono-agent context. A dependency graph between HTN tasks is built, that allows the replanning module to know which tasks need to be replanned in case of failure. Hence, they avoid recomputing well on-going parts of the plan. Considered data is adapted for re-planning processes, but they are not dealing with real robotic missions.

(Sotzing, Johnson, and Lane 2008) explicitly deal with knowledge representation and update in multiple AUV (Autonomous Underwater Vehicle) operations. Mission execution and multi-vehicle coordination is supported by BI-IMAPS (Blackboard Integrated Implicit Multi-Agent Planning Strategy). All vehicles have a complete copy of the BI-IMAPS plan, so the vehicles' actions can be predicted when there is no communication. When communication is anew available, robots refresh the knowledge coming from other vehicles. Communication management is a well known issue in multi-robot architectures. Furthermore, entities which are interacting must agree on the communication protocol, and the exchanged type of data. An implicit agreement about data is often considered in robotic team mission, but not formally described. When the same architecture is controlling all the robot team, (Tambe 1997) points out that *the more the team is flexible and robust, the higher communication load is required*. Thus, he emphasizes the importance of communication management in team mission execution, as this represents the most critical resource and may lead to a mission failure. In robotic space exploration missions, mainly because of communication constraints, it is impossible to consider a fine coordination of a set of vehicles from a mission control based on Earth. Part of this coordination must therefore be done on site, by the robotic agents themselves. Consequently, local communications between vehicles and a significant degree of autonomy in the vehicles' interactions will be assumed. Different space agencies (ESA, NASA...) already started to evoke future needs concerning standards and interoperability (Tramutola and Martelli 2010; Chien et al. 2006; Merri et al. 2002). Meanwhile communication standards and procedures are already used by (Kazz and Greenberg 2002), but at the equipement level.

In the literature of heterogeneous multi-robots missions, we can notice that protocol formalization is not satisfying. First, regarding protocols for plan execution and repair, ad-hoc interfaces (between planners and vehicles) are implicitly used to translate the mission plan into a language understandable by an autonomous vehicle. Plan repair is often considered from a planning point of view, not from the point of view of the generation process of the data that must be provided to the planner, and coming from heterogeneous sources. Besides, regarding protocols for multi-robot communication, a common assumption is made that all autonomous vehicles of a same team are implicitly working with the same standards.

For these reasons, we propose to formalize the common knowledge involved in (re)planning and executive phases. We describe first our knowledge representation, based on an ontology describing what the vehicles are able to achieve, and how interactions are managed. Afterwards, we present the use of an instantiated version of this ontology for mission planning, plan execution, and on-line plan repair.

Knowledge Representation

Ontologies in robotics

An ontology is a common representation of a specific domain that allows different individuals to share concepts and rich relations (Baclawski and Simeqi 2001).

In robotics, ontologies are used to specify and conceptualize a knowledge accepted by a community, using a formal description to be machine-readable, shareable (Sellami et al. 2011) and to reason over that knowledge to infer additional information (Schlenoff and Messina 2005). Ontologies offer significant interests to multi-agent systems such as interoperability (between agents and with other systems in heterogeneous environments), re-usability and support for multi-agent system development activities (Tran and Low 2008).

Ontologies must be used with care (Lortal, Dhouib, and Gérard 2011): an ontology (specification) must not be confused with a knowledge base (which actually includes knowledge). In the following, an *instantiated ontology* will refer to the knowledge base. Ontologies depend highly on their builders, and allow sharing of information between agents (Deplanques et al. 1996).

Ontologies are already being used in many different projects. In the context of Web Service the system of (Sirin et al. 2004) executes a plan computed with SHOP2 (Nau et al. 2003) over the web. It is able to execute information-providing Web Service during the planning process. We must note that they provide a sound and complete algorithm to translate OWL-S[2] service description into a SHOP2 domain. The Robot Earth European project (Waibel et al. 2011) aims at representing a world wide database repository where robots can share information about their experiences, with abstraction to their hardware specificities. But it is a starting project, without exploitable results yet, and it deals more about environment knowledge representation and sharing. In the Proteus project (Lortal, Dhouib, and Gérard 2011), complex ontologies are used for scientific knowledge transfer between different robotics communities. However, the developed ontology cannot be used directly for code generation and exploitation: authors have to perform semi-automatic transformation from the ontology to a UML representation. The ontology is also quite specific to their application scenarios problems. For space applications, the SWAMO NASA project (Witt et al. 2008) uses ontology as a prototyping method to provide standard interfaces to access different mission resources (sensors, agent capabilities...). In a similar approach, the A3ME (Herzog, Jacobi, and Buchmann 2008) ontology defines heterogeneous mobile devices, to allow communication interoperability. (Schlenoff and Messina 2005) has also worked on robots' capabilities representation in the context of urban search and rescue missions.

Those studies are very interesting and represent a starting point for our work, but the proposed ontologies are there at a lower level of knowledge representation. They focus more on the description of the capacities of mobile agents than on high level services representation for autonomous agents, as we aim to do. Because of these limitations, we choose to define a new ontology.

Formal robot description

The information we need to model in our ontology must take into consideration (1) robots involved in the mission and the information they are able to provide, (2) a description of the environment, and (3) a description of the mission goals. We must note that many ontologies already define services in the Web Service domain (e.g. OWL-S, or WSDL[3]). Our goal is to use a simple, effective ontology to support plan execution and repair. Different kinds of languages exist to model ontologies in computer readable text format. Our choice naturally falls on the most used in the domain of Artificial Intelligence which is the Web Ontology Language (OWL), defined by the World Wide Web Consortium[4] (W3C). OWL is an XML-based format for writing ontologies in Description Logic (DL). Many tools have been developed to design and manipulate ontologies. We have used Protégé[5] to design our ontology which is partially depicted in Fig. 1 and explained below.

To define this ontology, we were inspired by real robots execution management that we have been experimenting before and on the paradigm of programming by contract for the services that a robot can provide (Brugali and Scandurra 2009).

Robot Services Robots are classically defined by the services that they provide, and how agents can be interfaced with them (Brugali and Scandurra 2009). A `Service` is a task or an activity that a robot (or a software agent) can achieve in its environment (including what its sensors capabilities are able to get at a high level, e.g. target detection, rock analysis...). A service must implement a `Contract` (`SerContract`). It may need input parameters (`ConNeed`) to be executed and may return output parameters (`ConReturn`). It may be associated with specifications such as preconditions (`ConRequire`), postconditions (`ConEnsure`) or invariants (`ConInvariant`).

A `Service` is defined by its name (`SerName`) and a description (`SerDescription`). `SerAccess` describes how a distant entity (distant robot, team executor manager...) can call the service of the robot, to make the robot achieve it. `SerType` distinguishes *direct* services that can be directly called (goto action, take a picture, process data, etc.), and an execution result from them is expected, and *daemons*, which are running as background tasks and may return particular events (target detection, communication link broken, ...).

Robot Internal Variables A description of the knowledge manipulated by the robot is also required, all the more as these values are involved during the planning process. For example, if the robot has a GPS, it is a good assumption to have a $RobotPositionGPS$ variable. A variable, in the ontology, has classical entities: an identifier (`VarId`), a type (`VarType`) and optionally an informal text description (`VarDescription`). `VarGetter` and `VarSetter` are comparable to classical accessors in Object Oriented Programming, meaning the way to access the value of the variable, and a potential way to modify its value. The localization (place where the value is stored) of the variable (`VarLoc`) is defined, which is indirectly linked with the `VarInfluencers`. This last entity indicates which elements (vehicles, operators, variables...) may have an influence on the variable, and which elements must be observed during the execution process to detect particular evolution of the variable value. For instance, a robot "controls" its own position variable, other robots cannot "influence" this variable but can only read it.

Environment, Models and Relations Environment variables (areas, objects of interest, ...) and goals are also mod-

[2] `http://www.w3.org/Submission/OWL-S/`

[3] `http://www.w3.org/TR/wsdl`

[4] `http://www.w3.org/standards/techs/owl`

[5] `http://protege.stanford.edu/`

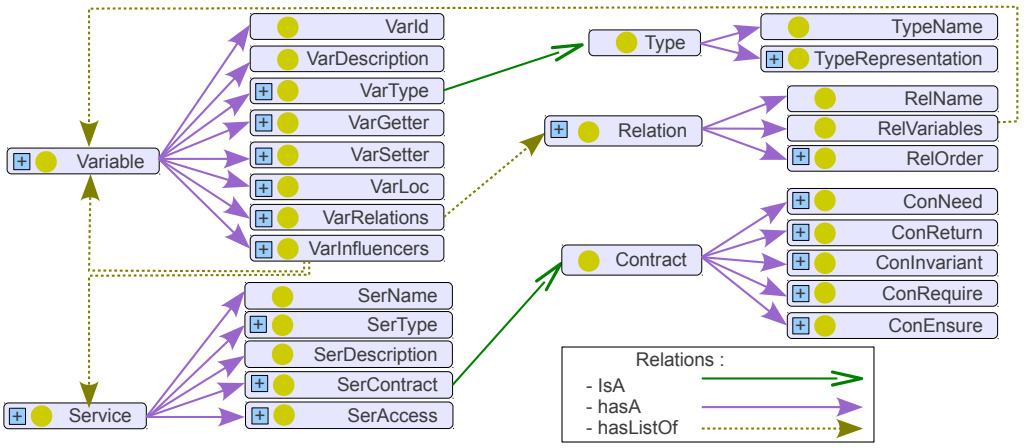

Figure 1: Partial view of the developed ontology, free adaptation from Protégé-Ontograf viewer

eled as `Variable` entities. In addition to traditional variable type, we use a particular type ($Model$) corresponding to robotic models: they allow to link some variables with external models, such as estimations of communication availability, estimations of the fuel consumption of a robot, etc.

`Relation` entities allow to interconnect variables when needed, and to add particular characteristics to them. For instance, a zone $z1$ can be a sub-zone of a zone $z2$, without creating new specific types for these variables.

With this ontology, we have a description of what the vehicles are able to achieve, how interactions are managed, and in what kind of environment they are dived into. Finally, this ontology regroups both planning and execution information.

Ontology in practice

The ontology described before is meant to be instantiated. The ontology provides the interface between both execution and plan-repair processes, and between the execution process and the vehicle's architecture.

In the following we present how to use the instantiated ontology, named KOPER (KnOwledge base for Planning, Execution and Repair) for (1) generation of planning model, (2) plan execution monitoring and (3) on-line repair. In each case, we discuss a generic use-case of the ontology, and then give a more concrete example of our own HTN-based architecture.

Planning Model Generation

We discuss here a mean to generate from the KOPER ontology the required information destined for a planner, which is then able to compute a mission plan.

Generic Planning Model Generating a complete planning model from an external data base without human intervention is far from being easy when confronted with real world missions, unless these real world missions are already fully described in a planning language. For that, we propose an automatic generation of the domain file from a KOPER instantiated ontology. While such a generation pattern can be generalized to almost all the planning languages, we illustrate this generation on the classical PDDL language (Drew McDermott et al. 1998). In PDDL, a planning model is made of a planning domain (that describes predicates and actions) and a planning problem (that instantiates domain variables, and defines the initial and goal states).

PDDL Domain: A PDDL domain is made of a set of $actions$. A PDDL action is described by a name, some parameters, a precondition and an effect. We can use Model to Model (M2M) transformation to generate the corresponding PDDL action from each `Service` of the instantiated ontology. `SerName` corresponds to the PDDL action name. `ConNeed` provides the PDDL action parameters information, `ConRequire` corresponds to the action preconditions, while `ConEnsures` and `ConInvariant` determine the action effects. Some `Variable` of the instantiated ontology can be involved in the domain generation, since they can be "influenced" by services. Listing 1 presents the instantiation of a *goto* Service provided by an instantiated $Vehicle$, an AAV (Autonomous Aerial Vehicle). The corresponding PDDL action is shown in listing 2, in which the $?v$ variable corresponds to a $Vehicle$.

PDDL Problem: A PDDL problem is made of a set of objects, an initial state and a goal. We use again a M2M transformation to generate the PDDL problem by considering entities `Variable` and `Relation`. The set of PDDL objects is generated from the `VarId` and the `VarType` of each `Variable`. The initial state description requires an exhaustive cover of the `Variable` values, accessed with combined `VarGetter` and `VarLoc` information. The `Variable` planning value is then formatted depending on the `VarType` documentation. The `Relation` entities are then interpreted, since written variables may be inter-linked. Finally, the goal is similarly generated, corresponding in our instantiated ontology to a particular `Variable`. Listing 3 presents a variable definition in the ontology and some relations, and listing 4 the generated PDDL problem.

HTN Planning Model in HiDDeN HiDDeN (Gateau, Lesire, and Barbier 2010) is a High level Distributed De-

```
<Service>
 <SerName>goto</SerName>
 <SerType>DirectTask</SerType>
 <SerDescription>
   The vehicle moves to the ?wptTo location
 </SerDescription>
 <SerContract>
  <ConNeed>
   <Arg>
    <ArgName>?wptFrom</ArgName>
    <ArgType>Waypoint</ArgType>
   </Arg>
   <Arg>
    <ArgName>?wptTo</ArgName>
    <ArgType>Waypoint</ArgType>
   </Arg>
  </ConNeed>
  <ConReturn>
   <Arg>
    <ArgName>?msgReturn</ArgName>
    <ArgType>SimpleReturn</ArgType>
   </Arg>
  </ConReturn>
  <ConRequire>
    <Cond>(this.pos == ?wptFrom)</Cond>
    <Cond>(this.st != this.onGround)</Cond>
  </ConRequire>
  <ConEnsure>
    <Cond>(this.pos == ?wptTo)</Cond>
    <Cond>(this.pos != ?wptFrom)</Cond>
  </ConEnsure>
  <ConInvariant>
   <Cond>(this.enoughFuel)</Cond>
  </ConInvariant>
 </SerContract>
 <SerAccess
  ComBase="Socket"
  OutPort="60100"
  ParamUsed="?wptTo"
  OutMsgFormat="SocketMsg('GOTO'+Waypoint)"
  InPort="60101"
  InMsgFormat="SocketMsg(SimpleReturn)"/>
</Service>
```

Listing 1: Example of possible "goto" service contract for an AAV in XML representation transformed from the OWL description ; the *this* keywords refers to the service owner

```
(:action goto
 :parameters (?wptFrom ?wptTo - Waypoint ?v
     - Vehicle)
 :duration (= ?duration (
    function_gotoDuration ?wptFrom ?wptTo ?
    v))
 :condition ((enoughFuel ?v)(position ?v ?
    wptFrom)(not (onGround ?v)))
 :effect (and((not(position ?v ?wptFrom))(
    position ?v ?wptTo))))
```

Listing 2: "goto action" in pddl style

```
<Variable>
 <VarId>aav1Position</VarId>
 <VarType>Waypoint</VarType>
 <VarGetter>Service::getGPSPoint</VarGetter>
 <VarSetter>none</VarSetter>
 <VarLoc>local</VarLoc>
 <VarRelations>
  <RelName>aav1PositionRelation</RelName>
  <RelVariables>
   <VarId>this</VarId>
  </RelVariables>
  <RelOrder>simple</RelOrder>
 </VarRelations>
 <VarInfluencers
  var="this"
  ser="Service::goto"/>
</Variable>
```

Listing 3: Example of a variable in XML representation transformed from the OWL description; the *this* keywords refers to the variable owner

```
(:objects
 aav1 - Vehicle
 aav1Position - Waypoint
 wpt2 - Waypoint)
(:init
 (aav1 at aav1Position))
(:goal
 (aav1 at wpt2))
```

Listing 4: Partial PDDL problem: $aav1Position$ is described in Listing 3. Variable $wpt2$ comes from similar OWL descriptions, alike a $goalVariable$ which precises that $aav1$ must reach waypoint $wpt2$; $aav1$ is an instantiated vehicle; a relation exists between $aav1$ and $aav1Position$ (`RelVariables` field in Listing 3) that leads to the generation of the initial state line.

cisioN layer that monitors execution and plan repair for a team of autonomous robots. The underlying planning model is based on HTNs (Nau et al. 2003). An HTN is a hierarchical set of abstract and elementary tasks. One or more methods are assigned to an abstract task and describe the way to achieve it, using other abstract or elementary tasks. The selection of a method is constrained by the fulfillment of preconditions. Theoretically, if the recipe provided by a method, picked out amongst the possible methods of the task, is followed, then the objectives of the task will be achieved. The interests of HTN are their flexibility, their hierarchical structure and their convenient modeling of human knowledge. Besides, multi-agent planners have been specifically developed to deal with HTN formalism (Dix et al. 2003).

Regarding our instantiated ontology, we can add the HTN structure (i.e. the abstract tasks corresponding to human expertise) either in the knowledge representation as "high level" `Service` entities, or directly in the generated planning model. We chose the second solution as human exper-

tise is in a sense more related to the planning model (an expert can add abstraction, but also domain-specific heuristics or constraints) than to robot or mission description.

One of the modeled missions is a mine hunting scenario involving an AAV and an AUV. The instantiation of the KOPER ontology for this scenario is composed of 18 robots' services. 34 variables are described (including goals and models), with 10 relations. The generated HTN domain contains 18 elementary tasks, increasing to 23 when we add HTN abstraction from a human expert. The generated HTN problem contains 26 objects, and 92 conditions in the initial state.

We have solved this problem using SHOP2, and our HTN planner. Both planners generate a plan containing 48 abstract tasks and 60 elementary tasks.

Plan Execution

After generating a planning model from the KOPER instantiated ontology, we can use an off-the-shelf planner to compute a mission plan. In this section, we describe how the KOPER ontology can assist in monitoring the plan execution, assuming that a mission plan has been computed.

Generic Plan Execution Plan execution consists in executing each action written in the plan by the concerned robot(s). Since robots have specific protocols for external interaction with them, the plan action must be translated according to this protocol, which describes how to make the robot execute the desired action. Thanks to the KOPER instantiated ontology, the action encountered by the execution manager is automatically mapped with this communication protocol, in particular with the `SerAccess`. In fact, `SerAccess` precises the type of communication mode, the associated configuration and also the data format and content that must be sent. When a robot executes an action, a report is expected to indicate whether the task has been completed or not. Once again the format and the manner are described in the KOPER instantiated ontology through `ConReturn` and `SerAccess`. The service contract (`SerContract`) allows failure detection means during the execution process. The execution manager may check that: (a) action's precondition (`ConRequire`) is true before executing, (b) action's actual effects are consistent with the action model (`ConEnsure`) and (c) action's invariants remain true during the execution.In fact, KOPER provides a mapping between execution variables and planning (logical) variables.

For instance, we take the action "goto wpt1 wpt2 aav1" from a PDDL planner output plan: AAV *aav1* must go from waypoint *wpt1* to *wpt2*. The execution manager finds in the KOPER instantiated ontology the corresponding service with the `SerName` "goto" (listing 1). It is then aware that communication with the local architecture is based on sockets (line 38), with a given port and a message format to send the execution order to the robot. The "Waypoint" parameter (from the `ConNeed` field) is passed to the robot, after getting its value through the `VarGetter` function. Finally, the plan execution process can send a socket message formatted into understandable data to the robot architecture. When this message has been sent, the execution manager waits for a "SimpleReturn" type message, on another socket, that corresponds to waiting for a boolean report (`true` if the robot has reached $wpt2$, `false` otherwise).

Daemon services are not triggered by the execution process: they are active all along the mission and can send information to the execution process according to the protocol described in the ontology. Such information can correspond to specific observations (e.g., a detected target) and most of the time will result in a plan adaptation or repair.

HTN-based Plan Execution in HiDDeN In HiDDeN, we directly use the HTN structure of the mission plan (hence resulting in an instantiated HTN, see (Gateau, Lesire, and Barbier 2010) for details) as the core model of plan execution monitoring.

HiDDeN is a distributed architecture that settles on the existing robots' architectures, takes care of what specific action has to be executed by which robot, and controls the coordination between robots (Fig.2). Each local supervisor is then able to communicate with all the rest of the team, depending on communication availability.

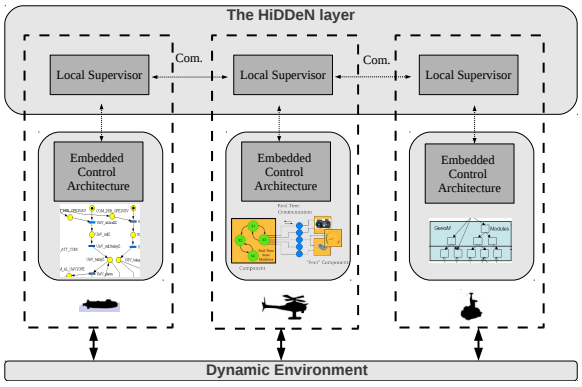

Figure 2: The HiDDeN layer monitoring a team of heterogeneous robots.

The main purpose of the local supervisor is to provide the action that the robot has to execute, depending on the local plan. The local HTN tree plan is followed in a depth first and left first manner. When an elementary task (a leaf) is reached, the corresponding action execution is requested to the local architecture (as explained in Generic Plan Execution). For instance, in Fig. 3, the root task to achieve is R. The execution manager enters into task $T1$, itself decomposed into $T2$. $E1$ must then be executed first by the robot. If this execution is successful (i.e. the report is `true`), $E2$ must then be executed. At the end of $E2$, $T1$ is considered as done. Afterwards, the execution manager enters into $T3$, and so on.

Failure detection in HiDDeN is based on monitoring actions' contracts (as described in Generic Plan Execution), analysing returned values after the execution ($ConReturn$), and timeouts events.[6].

[6]To avoid freezing, the execution engine arms a timer when run-

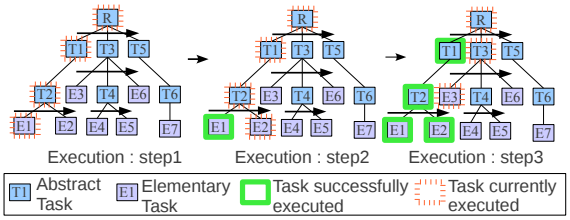

Figure 3: Execution of an instantiated HTN: the elementary execution order is $E_1, E_2, E_3, E_4, E_5, E_6, E_7$.

Regarding multi-robot coordination, HiDDeN plans are computed in order to ensure regular rendezvous between vehicles all along the mission. These rendezvous are 4D space (location and time) where two vehicles are able to communicate. During these rendezvous, the distributed supervisors of HiDDeN will update their team knowledge. This update is based on the KOPER instantiated ontology, that defines the team variables, their `varInfluencers` (which own the actual value of the data) and variable getters and setters (to access the data and store it).

Plan Repair

While executing the plan mission, the autonomous team will encounter failures. The team must then find an alternative plan to achieve the mission. For that, thanks to the KOPER instantiated ontology, an up-to-date planning model can be provided to an embedded planner, which can repair the current plan or compute a new plan.

Generic Plan Repair We consider here that plan repair is the responsibility of the execution process, in the sense that it consists in defining a new planning model corresponding to the part of the plan that needs to be repaired.

This process is hence very similar to the initial generation of planning domain and problem. However, variables' values have been modified during the execution process (e.g. the position of the vehicle has changed after a "goto" action). Updating these variable values is done through the KOPER instantiated ontology thanks to `VarLoc` and `VarGetter`.

The services list description can also change (e.g a service has an unexpected result and is "desactivated", or is not provided any more by a robot, a robot is out of service...), impacting then a different planning domain generation.

HTN Plan Repair in HiDDeN Plan repair in HiDDeN is globally done the same way as in Generic Plan Repair: by defining the local planning model to solve. However, we extensively use the HTN hierarchy in this process: once an elementary action failed, we first try to repair it (by defining a planning model corresponding to performing this action). If this repair succeeds, we replace the task and go on with execution. If the repair fails (either because no solution to the local problem can be found, or because the problem cannot even be defined due to unavailable data), HiDDeN goes up in the hierarchy and tries to repair the task just above the previous one (i.e. the abstract task containing the previous task), and so on until a repair plan is computed (Fig. 4).

As in our scenarios communication between the vehicles is not always available and is subject to disturbances, the KOPER instantiated ontology is very useful in providing the localization of variables that must be accessed to define the new planning problem. It limits communication to the required ones and allows to determine which communication links must be established to repair the plan and achieve the mission.

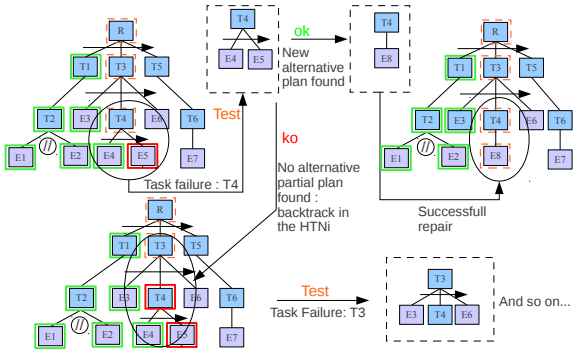

Figure 4: The repair process : elementary task E_5 fails, which implies a repair of task T_4 (top, left). If a new plan is available, the corresponding instanciated HTN branch is replaced (top, right); else, the first hierarchically superior task of T_4, here T_3, is repaired (bottom).

Conclusion

In this paper, we have defined an ontology, KOPER, to represent the necessary knowledge used during planning, plan execution and plan repair for a team of heterogeneous autonomous robots. KOPER is based on a description of robots' services with associated parameters and contracts, a description of variables with their access means and influencers, and a description of environment and robot models. KOPER holds at the same time information dedicated to the planning process and to the execution phase.

We have shown how an instantiation of KOPER can be used to (1) generate a planning model for off-line computation of an initial plan, (2) monitor the plan execution by inferring the execution protocol, and (3) help in defining a new planning problem for plan repair. We have discussed a generic use case of KOPER, and illustrated our personal experience using the HiDDeN architecture with concrete experiments on multi-robot missions. In this context, KOPER also helped us in managing necessary and sufficient communication in order to ensure the best possible execution of the mission.

References

[Baclawski and Simeqi 2001] Baclawski, K., and Simeqi, A. 2001. Toward Ontology-Based Component Composition. In *OOPSLA Workshop*.

[Brugali and Scandurra 2009] Brugali, D., and Scandurra, P.

ning an action. When the timer has expired, the action is interrupted and a failure is detected

2009. Component-based robotic engineering (Part I). *IEEE Robotics Automation Magazine* 16(4).

[Chaimowicz et al. 2001] Chaimowicz, L.; Sugar, T.; Kumar, V.; and Campos, M. 2001. An architecture for tightly coupled multi-robot cooperation. *ICRA*.

[Chien et al. 2006] Chien, S.; Doyle, R.; Davies, A.; Jonsson, A.; and Lorenz, R. 2006. The Future of AI in Space. *IEEE Intelligent Systems* 21(4).

[Deplanques et al. 1996] Deplanques, P.; Yriarte, L.; Zapata, R.; and Sallantin, J. 1996. An ontology for robot modeling and testing. In *CESA Symp. on Robotics and Cybernetics*.

[Dix et al. 2003] Dix, J.; Muñoz-Avila, H.; Nau, D.; and Zhang, L. 2003. IMPACTing SHOP: putting an AI planner into a multi-agent environment. *Annals of Mathematics and AI* 37(4).

[Doherty, Kvarnström, and Heintz 2009] Doherty, P.; Kvarnström, J.; and Heintz, F. 2009. A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. *JAAMAS* 19(3).

[Drew McDermott et al. 1998] Drew McDermott, M. G.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL: the planning domain definition language. Technical Report CVC TR-98-003, Yale Center for Computational Vision and Control.

[Fazil Ayan et al. 2007] Fazil Ayan, N.; Kuter, U.; Yaman, F.; and Goldman, R. 2007. HOTRiDE: hierarchical ordered task replanning in dynamic environments. In *ICAPS*.

[Gateau, Lesire, and Barbier 2010] Gateau, T.; Lesire, C.; and Barbier, M. 2010. Local plan execution and repair in a hierarchical structure of sub-teams of heterogeneous autonomous vehicles. In *ICAPS Doctoral Consortium*.

[Herzog, Jacobi, and Buchmann 2008] Herzog, A.; Jacobi, D.; and Buchmann, A. 2008. A3ME-an Agent-Based middleware approach for mixed mode environments. In *Mobile Ubiquitous Computing, Systems, Services and Technologies*.

[Kazz and Greenberg 2002] Kazz, G., and Greenberg, E. 2002. Mars Relay Operations: Application of the CCSDS Proximity-1 Space Data Link Protocol. Technical report, JPL - NASA.

[Lortal, Dhouib, and Gérard 2011] Lortal, G.; Dhouib, S.; and Gérard, S. 2011. Integrating ontological domain knowledge into a robotic DSL. In *MODELS*.

[McGann et al. 2008] McGann, C.; Py, F.; Rajan, K.; Thomas, H.; Henthorn, R.; and McEwen, R. 2008. A deliberative architecture for AUV control. In *ICRA*.

[Merri et al. 2002] Merri, M.; Melton, B.; Valera, S.; and Parkes, A. 2002. The ECSS Packet Utilization Standard and Its Support Tool. In *SpaceOps Conference*.

[Nau et al. 2003] Nau, D.; Au, T.-C.; Ilhami, O.; Kuter, U.; Murdock, W.; Wu, D.; and Yaman, F. 2003. SHOP2: an HTN planning system. *JAIR* 20.

[Paolucci, Shehory, and Sycara 2000] Paolucci, M.; Shehory, O.; and Sycara, K. 2000. Interleaving planning and execution in a multiagent team planning environment. *Linköping Elec. Articles in CIS* 5(18).

[Parker 1998] Parker, L. E. 1998. ALLIANCE: An Architecture for Fault Tolerant Multi-Robot Cooperation. *IEEE Trans. on Robotics and Automation* 14.

[Schlenoff and Messina 2005] Schlenoff, C., and Messina, E. 2005. A robot ontology for urban search and rescue. In *ACM workshop on Research in knowledge representation for autonomous systems*.

[Sellami et al. 2011] Sellami, Z.; Camps, V.; Aussenac-Gilles, N.; and Rougemaille, S. 2011. Ontology Co-construction with an Adaptive Multi-Agent System: Principles and Case-Study. *Knowledge Discovery, Knowlege Engineering and Knowledge Management*.

[Sirin et al. 2004] Sirin, E.; Parsia, B.; Wu, D.; Hendler, J.; and Nau, D. 2004. HTN Planning for Web Service Composition Using SHOP2. *Journal of Web Semantics*.

[Sotzing, Johnson, and Lane 2008] Sotzing, C. C.; Johnson, N.; and Lane, D. M. 2008. Improving multi-AUV coordination with hierarchical blackboard-based plan representation. In *PlanSIG*.

[Tambe 1997] Tambe, M. 1997. Towards flexible teamwork. *JAIR* 7.

[Teichteil-Königsbuch, Lesire, and Infantes 2011] Teichteil-Königsbuch, F.; Lesire, C.; and Infantes, G. 2011. A generic framework for anytime execution-driven planning in robotics. In *ICRA*.

[Tramutola and Martelli 2010] Tramutola, A., and Martelli, A. 2010. Beyond the Aurora Architecture for the new challenging applications. The Enhanced Avionics Architecture for the Exploration Missions. Technical report, Thales Alenia Space.

[Tran and Low 2008] Tran, Q., and Low, G. 2008. MOBMAS: A methodology for ontology-based multi-agent systems development. *Information and Software Technology* 50(7-8).

[Vidal et al. 2002] Vidal, R.; Shakernia, O.; Kim, H. J.; Shim, D. H.; and Sastry, S. 2002. Probabilistic pursuit-evasion games: Theory, implementation and experimental evaluation. *IEEE Trans. on Robotics and Automation* 18.

[Visentin 2007] Visentin, G. 2007. Autonomy in ESA Planetary Robotics Missions. Technical report, ESA.

[Waibel et al. 2011] Waibel, M.; Beetz, M.; Civera, J.; D'Andrea, R.; Elfring, J.; Galvez-Lopez, D.; Haussermann, K.; Janssen, R.; Montiel, J.; Perzylo, A.; Schiessle, B.; Tenorth, M.; Zweigle, O.; and van de Molengraft. 2011. RoboEarth-A World Wide Web for Robots. *IEEE Robotics Automation Magazine* 18.

[Witt et al. 2008] Witt, K. J.; Stanley, J.; Smithbauer, D.; Mandl, D.; Ly, V.; Underbrink, A.; and Metheny, M. 2008. Enabling Sensor Webs by Utilizing SWAMO for Autonomous Operations. In *NASA ESTC*.

Mixed-Initiative Procedure Generation and Modification - the Future of Unmanned Military Systems Guidance?

Ruben Strenzke, Nikolaus Theißing, and Axel Schulte

Universität der Bundeswehr Munich, Institute of Flight Systems

Werner-Heisenberg-Weg 39, 85577 Neubiberg, Germany

{ruben.strenzke, nikolaus.theissing, axel.schulte}@unibw.de

Abstract

In this article we explain our position concerning the requirements of decision-making, planning, and plan execution engines for future unmanned military systems. We justify why we think that the application, modification, and generation of procedures in mixed-initiative interaction seems the most promising approach to effective human-machine systems, and refer to past and preliminary work. The procedure thereby becomes the common representation for planning, execution, and more or less supervised learning. The system's behavior is then regulated by a slightly dynamic database of procedures and a very dynamic set of currently applied human-generated constraints.

Introduction

Whereas the term 'robot' can refer to wide range of systems with different maturity levels and different functionality demonstration backgrounds, an uninhabited military vehicle (UMV) has special requirements concerning safety and intervention possibilities for the human operator(s). However, there are multiple reasons for raising the UMVs' autonomy, e.g. to lower the taskload for the human operator(s) or to handle situations in which there is no communication link (Billman and Steinberg 2007). In the latter case, rules of engagement (RoE) have to be readily defined in order to guarantee purposeful vehicle behavior with low risk of damage and injury. The human operator shall have the overall responsibility over the human-machine system, and therefore he/she shall be able to actively set and modify the rules of engagement. The ability to follow and effectively maintain action plans as well as RoE can be reached by providing a) onboard systems for decision autonomy of the UMVs and in addition to that b) a decision support system for the human operator. This dual constellation is also explained in (Linegang et al. 2006). In principle, both proposed systems can be built with the same a-priori world models and decision engine, which is why we do not differentiate between the two in the remainder of this article.

This article is intended to explain our position concerning the requirements imposed by future unmanned military systems in the areas of planning, decision-making, plan (or procedure) execution engines, corresponding hybrid architectures, automated learning, and human-robot interaction. We justify why we think that the application, modification, and generation of procedures in mixed-initiative interaction seems the most promising approach to well-performing human-machine systems in the field of UMVs.

Planning and Decision-Making

A UMV that is able to make decisions and re-plan its actions in case of a data link loss can be seen as an autonomous system. However, as explained in the beginning, there are special requirements concerning safety and human operator intervention associated with UMVs. Possible methods of intervention are given in (Strenzke and Schulte 2012) and can be summarized as giving a set of constraints, which can be either hard or soft, to the UMV's decision engine. The essence is that, whatever the UMV autonomously decides or plans in the mentioned situation, it must consider the following constraints:

- Physical or technical hard constraints concerning the vehicle itself (e.g. min. airspeed, current fuel level)
- Phys./technical soft constraints (e.g. prefer short routes)
- Tactical soft constraints (e.g. prefer reconned routes)
- Hard constraints set by the human operator (e.g. "UMV one will in no case use corridor Alpha")
- Soft constraints set by the human operator (e.g. "UMV one should use corridor Charlie", i.e. prefer Charlie)

The UMV operator receives *mission constraints*, which state the main goals of the mission and some mission-specific regulations, from another entity (e.g. headquarter).

He/she then gives *execution constraints* to the (multi-)UMV system. The execution constraints will, in most cases, be more detailed than the mission constraints. Figure 1 shows symbolically how the execution constraints limit the UMV autonomy. In certain cases some mission constraints may also be loosened by the UMV operator, e.g. he/she may sacrifice one UMV on the battlefield, whereas the mission constraints include that this UMV should be safe and home at the end of the mission (see how the light grey box leaves the dark grey box in Figure 1).

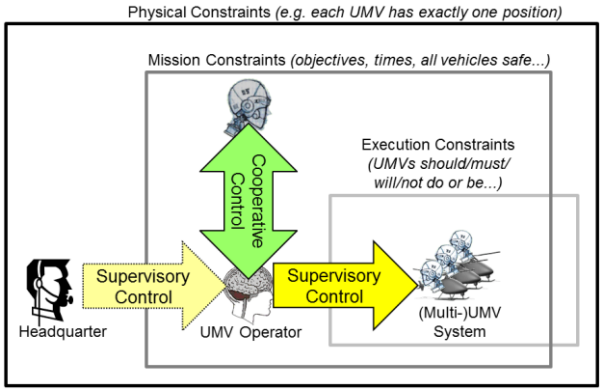

Figure 1: Different constraints limit the planning and decision space of the entities involved in UMV missions

Plan Execution and Procedures

After a mission plan has been agreed upon by the UMV system and the human operator, it has to be executed by the UMV system and the human operator's role becomes more passive as a monitor (supervisory control paradigm, see above). However, in a real-world environment with uncertainties concerning the distribution and behavior of mission-relevant objects and uncertainties concerning the success of action execution, reporting all unexpected high- and low-level events to the human operator may easily lead to cognitive overload. Therefore, certain events should be retained and resolved automatically by the UMV system. In order to keep the human in the loop of understanding the situation as well as the system's decisions, this automated event handling has to be common sense between the system and the human operator.

One way to accomplish this is the use of predefined *procedures*, i.e. schematic plans which may contain conditional branches as well as loops. If the number of procedures in the procedure database is small enough or if there is only a small set of frequently used, the operator is supposed to be able to know the procedures to an extent sufficient to understand and anticipate the system behavior in contingency situations.

Figure 2 illustrates the concept of procedures by the example of an Uninhabited Aerial Vehicle (UAV) flight

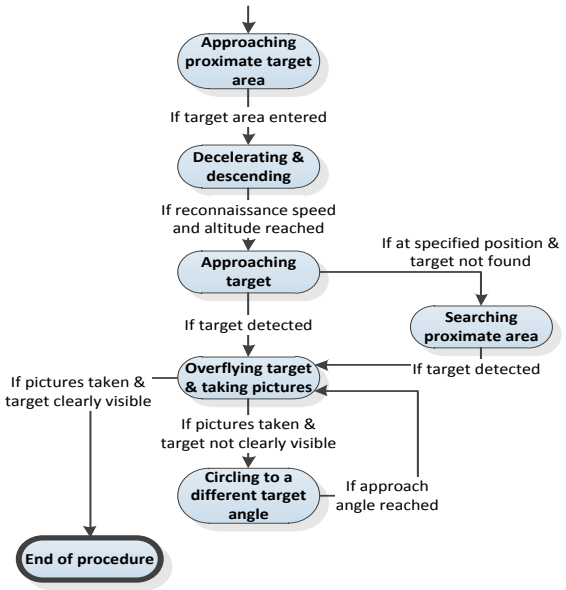

Figure 2: Example of a procedure with conditional branching and loops (UAV to recon stationary target)

mission. There, the procedure "Reconnoiter stationary target" is defined. It specifies the steps involved in carrying out a reconnaissance order. Additionally, it defines the reactions to various possible outcomes of each step. For example, the procedure has to be continued in a different way if a reconnaissance picture has been taken successfully than in the case that the target was not properly visible from the camera angle.

Procedure-Based Hybrid Architectures

When combining the requirements stated in the last two chapters, i.e. deliberative planning and procedural execution, we end up with a hybrid system architecture, which is centered on a procedure database. Examples for such systems are OpenPRS (Open Procedural Reasoning System) (Ingrand and Despouys 2001) or the 3T (Three Tier) architecture (Bonasso et al. 1997). The three layers in such architectures can be compared to Rasmussen's scheme of human performance (Rasmussen 1983). On the lowest of its three levels (*skill-based behavior*), the human performs unconscious control subroutines. In the center (*rule-based behavior*), signs are recognized and rules (procedures) are triggered depending on the current state and task. These procedures, which sequence the mentioned control subroutines,

> […] may have been derived empirically during previous occasions, communicated from other persons' know-how [...], or it may be prepared on occasion by conscious problem-solving and planning. […] During unfamiliar situations, faced with an environment for which no know-how or rules for control are available [...], the control of performance must move to a higher

conceptual level, in which performance is goal-controlled and *knowledge-based*. (Rasmussen 1983)

By using such procedure-based hybrid architecture for a UMV planning and execution engine, the *procedure* constitutes the common vocabulary between the human operator and the UMV system, which shall keep the operator in the loop and his task load low, as described before.

Our definition of a procedure bears resemblance to the so-called "plays" in the Playbook Approach (Miller et al, 2004), which in contrast lacks the possibility of on-line procedure definitions. Instead, we suggest providing a manual as well as a mixed-initiative in-flight creation and modification of procedures. In addition to that, we see procedures not only in the mission-level context (i.e. procedures which can span multiple UMVs and are designed to fulfill goals or parts of the mission), but also in the context of solving low-level problems like a technical failure onboard of a UMV.

Interestingly, (Frank, 2010) shows how humans are guided through manual procedures by automation support. This constellation is due to the even higher criticality in manned space missions. In our case, it will rather be the other way round, i.e. the human operator supports the automation to adapt its procedures to situations unprecedented by the system designers.

Automated Learning of Procedures

It is important to note that in this model, new procedures may be learnt by (low-level) planning, i.e. reasoning about a problem that is not similar to any previously encountered and solved problem. Using this functionality for UMV systems may greatly improve their performance. This idea is not at all new (Sussman 1973). Today there are non-deterministic (Cimatti et al. 2003) and probabilistic approaches to generate contingency plans either for remedy or prevention of problems, i.e. procedures containing conditional branches and loops. However, *automated learning of procedures may entail risks* in case of world model limitations, e.g. missing side-effect descriptions for some actions may lead to system damage when executing certain action combinations. This problem does not so much apply to the learning of macro procedures which merely consist of the combination multiple already defined procedures by high-level planning (i.e. treating whole procedures as planning operators).

Human-Robot Interaction

UMV behavior is defined by its more or less static *procedure database* and the *constraint set*, which is depending on the mission goals, the situation development, and preferences of the human operator. In this chapter, we explain our approach to human-robot interaction, which comprises

- *mixed-initiative planning* and *mixed-initiative plan execution* – two independent concepts that can benefit from each other when combined to *mixed-initiative operation* (Strenzke and Schulte 2012),
- *mixed-initiative constraint definition*, designed to support mixed-initiative plan execution, as well as
- *mixed-initiative procedure generation and modification*.

Mixed-Initiative Planning

As depicted in figure 1, the relationship between the human operator and the UMV system (white robot-heads) can be described as *supervisory control* (Sheridan 1992), which means that the operator first *plans* an operation (or mission) in his own mind, then *teaches* (in our case gives constraints to) the system, which then plans the mission. From then on he/she *monitors* the execution of the mission, and *intervenes* (i.e. re-teaches) if necessary. In contrast to this, the relationship between the human operator and the decision support system (or assistant system, grey robot-head) is of more cooperative nature, therefore called *cooperative control*. It knows the execution constraints, but it is not bound to those. Thereby, it is able to support the UMV operator in his job of defining and modifying the planner constraints, resulting in mixed-initiative planning. This setup has already been prototypically implemented (Strenzke and Schulte 2011) and evaluated in human-in-the-loop experiments (Strenzke and Schulte 2012).

Mixed-Initiative Plan Execution

Mixed-initiative plan execution means that in certain cases the human operator is somehow involved in triggering critical actions, as in (Cummings and Mitchell 2005), and in changing the plan in case of disturbing unforeseen events. In our concept of *mixed-initiative operation* (Strenzke and Schulte 2012), we propose to re-use the constraints defined by the human operator during the mixed-initiative planning process for later decisions during mission execution. These constraints then define the levels (degrees) of automation (Sheridan 1992), which define human operator involvement in system decisions.

Mixed-Initiative Constraint Definition

The problem with the mixed-initiative operation approach is that in some cases the UMV operator might only give few constraints to the system and nevertheless be satisfied with the resulting plan, thereby revealing only a small part of his overall intentions (i.e. actions that should take place, actions that should not take place). If this happens, the system should be able to infer about the human's intent and generate relevant contingency cases. It can then ask the operator for his recommendations or propose constraints

on its own in order to gather more human-approved constraints, which ease automated or mixed-initiative decision making before time-critical problems arise during mission execution. This field of work seems yet unexplored.

As an example, the human operator defines the constraints that UMV1 should not use corridor1 for ingress and UMV2 should not use corridor1 as well. The system is then able to infer either by analogy that the question if UMV3 also should not use corridor1 remains open. Of course, the system is able to decide by itself which corridor UMV3 will use, but as long as the operator's workload is low - which has to be measured or estimated (Donath et al. 2010) - , it makes sense to clarify this question. However, in the same situation, the system could infer that in a reconnaissance mission it makes sense to cover as many corridors as possible. This would lead to the question if UMV3 should do use corridor1, since corridor1 may remain uncovered.

Mixed-Initiative Procedure Generation and Modification

In addition to the execution constraints, the human operator should be able to generate new and modify existing low-level or high-level (macro) procedures for the UMV system, in order to master unforeseen problems during mission execution. However, in high task load situations, computer-assistance will be necessary. As explained before, it is risky to create or modify procedures completely automatically. The cooperation of human and machine during procedure generation or modification therefore seems to be the concept of choice for future UMV guidance approaches. After finding out that there is or will be a heat problem with the standard procedure in a new environment, the computer may recommend inserting the conditional action '*decrease-speed*' (with the condition '*outside-temperature > x*') before the lengthy search task and an *'increase-speed'* before the critical overfly. The problem may be detected by projecting future states during procedure simulation. Of course, the world model of the low-level planner must have sufficient detail in order to come to this conclusion, i.e., for this example, include the temperature variables. Then, whether the procedure is modified accordingly, is the human operator's responsibility. If the procedure is modified, the next time the system automatically decreases its speed in case it is too hot, without having to ask the human operator for permission, thereby lowering the task load and raising safety.

Human-Machine Interface

In the first place, to obtain system behavior predictable for the human operator, the human-machine interface should be mainly graphically-based (e.g. as in figure 2). However, there will be at least supporting dialogs for mixed-initiative interaction, which can be realized as natural language dialogs, as described by (Bonasso, 2002; Allen et al., 2007).

References

Allen, J. ; Chambers, N.; Ferguson, G.; Galescu, L.; Jung, H.; Swift, M.; and Taysom, W. 2007. PLOW: A Collaborative Task Learning Agent. In: *Proceedings of the National Conference on Artificial Intelligence* vol. 22 no. 2.

Billman, L., and Steinberg, M. 2007. Human System Performance Metrics for Evaluation of Mixed-Initiative Heterogeneous Autonomous Systems. In *Proc. of the Performance Metrics for Intelligent Systems (PerMIS) Workshop*, Gaithersburg, MD.

Bonasso, P. 2002. Using Discourse to Modify Control Procedures. In: *Proceedings of the AAAI Fall Symposium on Human Robot Interaction*, 9-14.

Bonasso, R. P.; Kortenkamp D.; Miller D. P.; and Slack, M. 1997. Experiences with an Architecture for Intelligent, Reactive Agents. In *Journal of Experimental and Theoretical Artificial Intelligence* 9: 237–256.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, Strong, and Strong Cyclic Planning via Symbolic Model Checking. In *Artificial Intelligence* 147(1-2): 35–84.

Cummings, M. L., and Mitchell, P. J. 2005. Managing Multiple UAVs through a Timeline Display. In *Proc. of AIAA Info Tech.*

Donath, D.; Rauschert, A.; and Schulte, A. 2010. Cognitive Assistant System Concept for Multi-UAV Guidance using Human Operator Behaviour Models. In *Proceedings of HUMOUS'10* (Humans Operating Unmanned Systems). Toulouse, France.

Frank, J. 2010. When Plans Are Executed by Mice and Men. In *Proceedings of IEEE Aerospace Conference*, 1–14.

Ingrand, F. F., and Despouys, O. 2001. Extending Procedural Reasoning toward Robot Actions Planning. In *Proceedings of IEEE International Conference Robotics and Automation* (ICRA), 9–14.

Linegang, M. P.; Stoner H. A.; Patterson, M. J.; Seppelt, B. D.; Hoffman, J. D.; Crittendon, Z. B.; and Lee, J. D. 2006. Human-Automation Collaboration in Dynamic Mission Planning: A Challenge Requiring an Ecological Approach. In *Proceedings of the HFES Annual Meeting* vol. 50 no. 23, 2482–2486.

Miller, C.; Goldman, R.; Funk, H.; Wu, P.; and Pate, B. 2004. A Playbook Approach to Variable Autonomy Control: Application for Control of Multiple, Heterogeneous Unmanned Air Vehicles. In *Annual Meeting of the American Helicopter Society*.

Rasmussen, J. 1983. Skills, Rules, and Knowledge; Signals, Signs, and Symbols, and Other Distinctions in Human Performance Models. *IEEE Transactions on Systems, Man, and Cybernetics* 13(3): 257–266.

Sheridan, T. B. 1992. *Telerobotics, Automation and Human Supervisory Control*. Cambridge, MA: MIT Press.

Strenzke, R., and Schulte, A. 2011. The MMP: A Mixed-Initiative Mission Planning System for the Multi-Aircraft Domain. In *Proccedings of the Scheduling and Planning Application woRKshop (SPARK'11)*, 74–81. Freiburg, Germany.

Strenzke, R., and Schulte, A. 2012. Design and Evaluation of a System for Mixed-Initiative Operation. *Acta Futura* 5: 83–97.

Sussman, G. J. 1973. A Computational Model of Skill Acquisition, Technical Report AITR-297, MIT AI Lab.

A Proposal For A Global Task Planning Architecture Using the RoboEarth Cloud Based Framework

Rob Janssen and **René van de Molengraft** and **Maarten Steinbuch**
Department of Mechanical Engineering
Den Dolech 2
5600 MB Eindhoven
the Netherlands

Daniel Di Marco and **Oliver Zweigle** and **Paul Levi**
Department of Image Understanding
Universitätsstrasse 38
70569 Stuttgart
Germany

Abstract

As robotic systems become more and more capable of assisting in human domains, methods are sought to compose robot executable plans from abstract human instructions. To cope with the semantically rich and highly expressive nature of human instructions, Hierarchical Task Network planning is often being employed along with domain knowledge to solve planning problems in a pragmatic way. Commonly, the domain knowledge is specific to the planning problem at hand, impeding re-use. Therefore this paper conceptualizes a global planning architecture, based on the worldwide accessible RoboEarth cloud framework. This architecture allows environmental state inference and plan monitoring on a global level. To enable plan re-use for future requests, the RoboEarth action language has been adapted to allow semantic matching of robot capabilities with previously composed plans.

Introduction

During the last couple of years a lot of effort has been put in the integration of assistive systems in human-oriented domains. One of the main reasons, is that robots are well-suited to execute tasks that are either dangerous, strenuous or simply too boring for humans to be performed.

A major advancement in integrating robotic helpers in human environments would be the ability to autonomously plan and execute tasks that are described in a highly expressive, symbolic way. Symbolic planning methods that make use of domain knowledge have been shown to be quite successful in real-world applications (Au, Kuter, and Nau 2005).

For plan construction, these planning systems typically require to infer the current state of the world. For large environments however, this inference might be difficult or even impossible to accomplish if the planning system (usually a personal robot or household application) has no awareness beyond its own perceptual range. Examples can be found for instance in hospitals, where patients might change their locations, or in office environments, where doors close, elevators become occupied or other robots might require priority over certain appliances. In the RoboEarth project (Waibel et al. 2011), a global representation of the world, i.e. a *global world model*, is proposed which is continuously updated by all systems operating in that specific environment. If something changes in the environment relevant to the executing robot, a new plan can be directly provided, without having to wait until the robot reveals the discrepancy itself.

Planning robot tasks globally on a database instead of locally on an isolated system also brings in the possibility to re-use earlier constructed plans in future plan requests, hereby speeding up the search and enabling autonomous generation and memorization of task descriptions between robots. This paper will therefore conceptualize a planning architecture that allows to

- compose plans based on a globally and continuously updated view of the world,
- re-use full or partial plans that were constructed during previous planning requests.

In addition, a formal language describing a taxonomy of actions will be adopted, which will be expanded in this paper to be used in a state-of-the-art planning component. Plan refinement will be based on this taxonomy, where suitable decompositions will be based on a robot's unique capabilities.

Related work

Service robots are nowadays provided with countless possibilities to share the knowledge they have gained during planning or execution of earlier tasks. With the arrival of common machine-readable web languages (Yu 2007; Group 2009), open standardized component interfaces (ORIN, OROCOS, ORCA, ROS), task-centered service robot domain ontologies (Kang et al. 2009) and task-attribute related databases (Shilane et al. 2004; Waibel et al. 2011), there are now multiple ways to share and re-use task related knowledge between systems operating in this domain.

In the *Rosetta* project (Björkelund et al. 2011) a knowledge framework for sharing information of robot skills on a semantic level was conceptualized that allows to share formal knowledge between robots. To convert plant data into a formal representation, extensions were made

on *AutomationML* which allows this information to be stored as *RDF* triples on the semantic web. Exploiting the semantic web as means to share knowledge between robots has been done in the *Proteus* project (Lortal, Dhouib, and Gérard 2011), where the focus is to integrate ontological knowledge into a common robot Domain Specific Language (DSL). Because no concrete effort has been made yet to elaborate specifically on the descriptions of robot tasks, the RoboEarth action language (Tenorth et al. 2012) has been developed to allow a suitable, hierarchical representation of task related knowledge in this domain.

Because human task instructions usually only provide information on an abstract high level, decomposition of these instructions into robot executable primitives requires to solve a *planning* problem. *Hierarchical Task Network* (HTN) planning (Ghallab, Nau, and Traverso 2004) has been proposed to decompose high level plans into sub-plans (*methods*) and eventually into executable *operators*.

There exists a large amount of work on the integration of HTN planning methods with OWL descriptions for compositing web services. In (Hartanto and Hertzberg 2008) a method is proposed to extract a reduced HTN planning domain from an OWL encoded knowledge base. The concept described in this article is similar to a method proposed in (Sirin et al. 2004), where HTN planning was applied to OWL-S description logics after converting them to the SHOP2 domain.

RoboEarth Planning Architecture

The work described in this article presents a first proposal for the RoboEarth planning architecture, for which a schematic visualization is given in figure 1. On the RoboEarth Cloud Framework, a SHOP2 planner (Nau et al. 2003), is implemented that requires a triple (S_0, T, D) to generate plan P. When the robot receives a user instruction, it queries the RoboEarth cloud framework for an executable plan. With this query, the abstract user instruction is uploaded and directed in binary format to an intelligent processing service on the RoboEarth database, where it is parsed into a set of partially ordered set of tasks describing a SHOP2 problem T (Kemke 2006).

To derive the initial state of the world S_0, a global reasoning and inference system (Tenorth and Beetz 2009) is adopted, that is capable of deriving predicates describing the current state of the world. This derivation is based on two knowledge bases,

- a global world model (Elfring et al. 2011), that continuously tracks the state of the world by receiving updates from all connected robots (e.g. where a certain cup is located, or what floor an elevator is currently at),
- a global knowledge base, consisting of *facts* (e.g. if a cup can be used to transport liquids), and *computables* (Tenorth and Beetz 2009) that can be used to derive new facts (e.g. if a cup is on top of a table, based on their geometric relation),

A concept of this combined tracking, reasoning and inference system is described in (Waibel et al. 2011).

The SHOP2 planning domain D is constructed by a capability matching component, that composes usable sets of *methods* and *operators* based on a semantic description of the robot's capabilities. Details of this semantic description language (SRDL) are described in (Kunze, Roehm, and Beetz 2011).

After construction, the composed plan P is sent as a response to the robot through the SeRQL RDF query interface (Prud'hommeaux and Seaborne 2008). In parallel, the composed plan is also fed back to the RoboEarth compound plan database to be used as a sub-task in future planning requests.

To execute the received plan on the robot, the knowledge-driven execution engine CRAM (Beetz, Mösenlechner, and Tenorth 2010) is adopted. This execution engine allows to execute highly expressive, symbolic plans for which the specific plan parameters are derived at run-time. This parametrization is achieved by a reasoning and inference system running locally on the robot. This inference component decides on whether to use the robot's local knowledge base, e.g. for visual-servoing purposes, or if the global RoboEarth global knowledge base should be consulted, for instance if the numerical coordinates for an imperceivable navigational goal in another part of the building needs to be inferred, or if a proper set of grasp points for a locally unknown object are required.

The CRAM engine also allows to pre-define error handling methods to notify the RoboEarth database if a certain part of the plan cannot be performed and another plan is required. To compose a new plan, a different refinement will be selected in the planning process, hereby excluding the part of the plan that could not be accomplished. Methods for selecting preferable refinement strategies are described in (Tsuneto et al. 2007) and (Sohrabi, Baier, and McIlraith 2009).

It is also possible for the RoboEarth database to pre-empt the currently executed plan on the robot. This will happen, if the global reasoning component on the database discovers a relevant change in the initial state of the world that was assumed during plan construction time. An example could be that another robot discovers an elevator becoming unavailable, or the displacement of a task relevant person to another location.

RoboEarth Action Language

The RoboEarth action language (Tenorth et al. 2012) aims to be a semantic representation language that robots can use to autonomously exchange information among each other. It is based on the W3C-standardized Web Ontology Language (Group 2009), more specifically on the description logics variant OWL-DL. The language allows a formal, machine-readable specification of actions, objects, locations and also more abstract entities like unit types and robot capabilities.

Complex actions are stored in separate OWL files and are called action recipes (Di Marco et al. 2012). They are, similar to HTNs, constructed by compositing primitive actions and other compound actions. In contrast to classical HTN planning methods, the capability matching system on

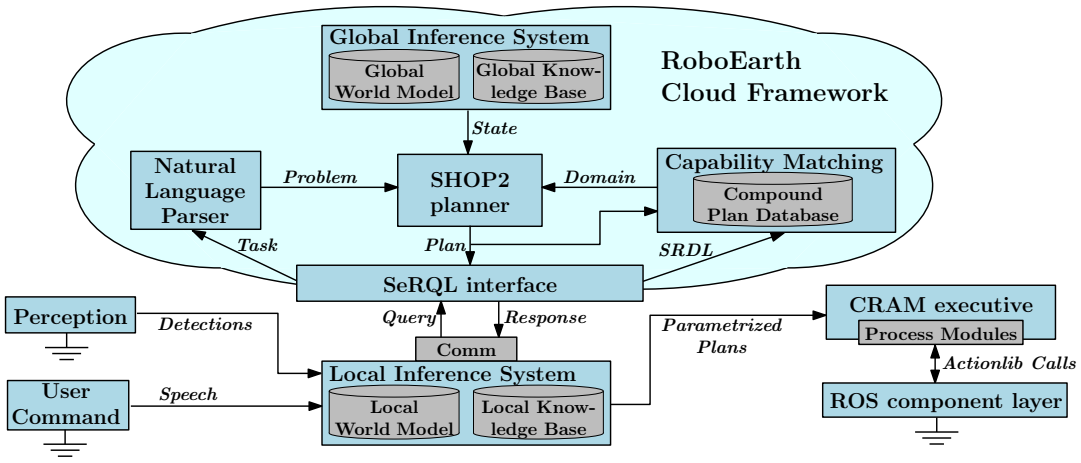

Figure 1: Proposed RoboEarth planning architecture.

the RoboEarth cloud selects the available actions suitable for execution by matching the capabilities of the respective robot platform with the capability requirements of the action. Additionally, it also checks whether the robot can be provided with missing task-related components, e.g. an occupancy grid map for navigation (Waibel et al. 2011).

Because the RoboEarth language was not designed for the purpose of domain-independant task planning, but with the idea of exchanging plan descriptions between different robot platforms in mind, the language has to be extended. In the following subsections these extensions will be presented.

Grouping methods by action hierarchy

HTN planning requires different methods that achieve the same task to be grouped together. In classical approaches, e.g. as implemented by the popular SHOP2 planner (Nau et al. 2003), methods implementing the same goal for different contexts are grouped by having the same name.

An alternative to this concept is to make the task the parent node of the methods implementing the goal. The required parameters for methods need to be stored as OWL properties in the parent node, because all methods implementing a task require the same parameters.

Action effects and preconditions

An HTN planning system synthesizes complex plans from a set of basic actions. For this, it requires a way to project the effects of these actions to a world state, hereby taking the imposed preconditions of each action into account. A common solution in planning literature to this problem is the use of predicates for describing symbolic changes in the world state for action preconditions and effects (Ghallab, Nau, and Traverso 2004). Thus, the extensions required to our language for describing actions in this work are: Action preconditions (for both methods and operators) and Action effects (for operators). Both can be added to the language as additional OWL object properties.

As the main purpose of the project is to enable different robot platforms to re-use action recipes, this opens the question of how to ensure consistent usage of predicates for describing the effects and preconditions of basic actions. Or formulated in terms of HTN planning: how can we describe different sets of operators (one set for each robot platform) such that they can be used by the same set of methods?

One solution would be to provide an additional ontology predefining these predicates and necessary logical operators like *and*, *or*, *not*, etc. Even though there are multiple standards available for describing first order logic formulas in OWL ontologies (as e.g. RIF[1]) we choose to provide our own. This primarily has two reasons:

- The knowledge representation semantics commonly chosen for description logics and thus OWL-DL make use of the *open world assumption*, i.e. facts not explicitly provable are assumed to be unknown. Symbolic planners usually implement the *closed world assumption* and consider facts that are not provable by the provided knowledge to be false.
- The mentioned standards focus on extending OWL-DL rules working on the knowledge base with methods of logic programming, whereas our goal is merely to describe a set of changes to a set of predicates. Higher expressivity comes at the cost of computational power for planning, so we tried to reduce the expressivity of the language as far as possible.

Capability matching and domain size reduction

As the number of methods in the intended system can be expected to be large, it is important to reduce the complexity of the planning search by removing methods that are not relevant to the task under consideration. We can achieve a first reduction of size by including solely the methods and predicates required by the tasks, using the method proposed in (Hartanto and Hertzberg 2008).

As was described in (Tenorth et al. 2012), the current RoboEarth system is able to reason about the requirements

[1] http://www.w3.org/TR/2010/REC-rif-core-20100622/

```
(:- (hasAllRequiredComponents ?robot ?cap)
    (or (not (capDependsOnCompRec ?cap ?comp))
        (hasComponent ?robot ?comp)))

(:- (capDependsOnCompRec ?cap ?comp)
    (or (capDependsOnComp ?cap ?comp)
        (and (capDependsOnCap ?cap ?cap2)
         (capDependsOnCompRec ?cap2 ?comp))
        (and (capProvidesCap ?cap ?provcap)
         (capDependsOnCompRec ?provcap ?comp))))

(:operator (!download-comp ?robot ?comp)
   ((downloadableComp ?comp)
    (not (hasComponent ?robot ?comp)))
   (())
   ((hasComponent ?robot ?comp)))

(:method (prepareCap ?robot ?cap)
  ((capDependsOnCompRec ?cap ?comp)
   (not (hasComponent ?robot ?comp)))
  ((!download-comp ?robot ?comp)
   (prepareCap ?robot ?cap))

  ((hasAllRequiredComponents ?robot ?cap)
   ())) ;; NOP
```

Figure 2: SHOP2 domain for capability matching

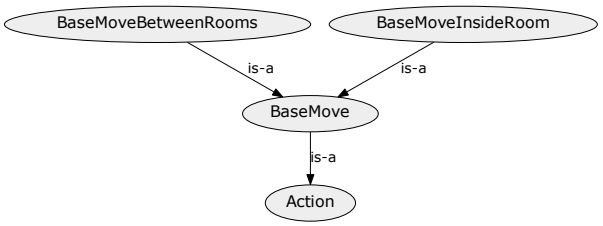

Figure 3: Example task hierarchy

of actions and capabilities of robots to find out whether an action can be executed by a robot platform, and which downloadable components would be required to do so[2]. Currently, this is done by several Prolog rules in the knowledge processing framework KnowRob (Tenorth and Beetz 2009). For a more coherent system in the sense of this work, we could transform the capability matching process into a HTN planning problem, where we define the *download-comp* operator and a recursive *prepareCap* method (fig. 2). An example is given in figure 3: The task under consideration is BaseMovement, which can be achieved by the methods BaseMoveInsideRoom and BaseMoveBetweenRooms, respectively. Assume for the sake of the example that BaseMoveInsideRoom implements a simple operator call to a 2D navigation method (MoveBasePrimitive), while BaseMoveBetweenRooms is a compound task incorporating sub-tasks for detecting and opening closed doors. In classical HTN notation, the methods could be described as shown in figure 4 (assume a sequential ordering for the subtasks).

[2]SRDL description can include additional information about robot capabilities, but currently only the mentioned capability checking is used.

BaseMoveBetweenRooms

task:	BaseMovement$(from, to)$
precond:	$inRoom(from, room1)$ $\wedge$
	$inRoom(to, room2)$ $\wedge$
	$doorConnecting(door, room1, room2)$
subtasks:	$\langle$BaseMovement$(from, door1)$,
	DoorOpen$(door1)$,
	NavigateThroughDoor$(door1)$,
	BaseMovement$(door1, to)\rangle$

BaseMoveInsideRoom

task:	BaseMovement$(from, to)$
precond:	$inRoom(from, room)$ $\wedge$
	$inRoom(to, room)$
subtasks:	$\langle$MoveBasePrimitive$(from, to)\rangle$

Figure 4: Example methods in HTN syntax

To generate a SHOP2 planning problem, the first step would be to collect all action recipes that achieve the BaseMovement task and translate them into the planner's syntax. We also need to retrieve recursively all the methods that are used in any BaseMovement methods, in this case DoorOpen and NavigateThroughDoor. We then can apply the capability matching process mentioned before on each of the methods retrieved so far to further decrease the number of methods and to retrieve missing data from the database.

Having collected these, the current state of the world has to be formulated as a set of predicates. Again, it is sufficient to collect predicates that are referenced by any of the methods remaining after the previous step. Finally, we can apply the SHOP2 planner on the reduced method set and world state description and create a sequential CRAM plan from the result, provided that planning succeeds.

Discussion & Future Work

We proposed a way to combine HTN planning with a global world model to enable robots to share task descriptions and synthesize plans.

An important possible improvement relates to the well-known lack of flexibility we have to accept when we create a static plan based on a snapshot of the world state. We intend to investigate ways on how to improve on that by integrating planning and execution more tightly, e.g. as proposed in (Kaelbling and Lozano-Perez 2011) and how to explicitly deal with unforeseen changes in the assumed state of the world state (Au, Kuter, and Nau 2005) (Ayan et al. 2007), (Warfield et al. 2007), (Keller, Eyerich, and Nebel 2010)

Acknowledgements

The research leading to these results has received funding from the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement number 248942 RoboEarth.

References

Au, T.; Kuter, U.; and Nau, D. 2005. Web service composition with volatile information. In *International Semantic Web Conference*, pp 52–66.

Ayan, N.; Kuter, U.; Yaman, F.; and Goldman, R. 2007. Hotride: Hierarchical ordered task replanning in dynamic environments. In *Proceedings of the ICAPS-07 Workshop on Planning and Plan Execution for Real-World Systems.*

Beetz, M.; Mösenlechner, L.; and Tenorth, M. 2010. CRAM - A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments. In *International Conference on Intelligent Robots and Systems (IROS)*.

Björkelund, A.; Malec, J.; Nilsson, K.; and Nugues, P. 2011. Knowledge and skill representations for robotized production. In *International Federation of Automatic Control (IFAC)*, pp 8999–9004.

Di Marco, D.; Tenorth, M.; Häussermann, K.; Zweigle, O.; and Levi, P. 2012. Roboearth action recipe execution. In *IAS Intelligent Autonomous Systems*. submitted.

Elfring, J.; van de Molengraft, M. J. G.; Janssen, R. J. M.; and Steinbuch, M. 2011. Two level world modeling for cooperating robots using a multiple hypotheses filter. In *International Conference on Robotics and Automation (ICRA)*, pp 815–820.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated planning - theory and practice*. Elsevier.

Group, W. O. W. 2009. Owl 2 web ontology language document overview. W3c recommendation, W3C.

Hartanto, R., and Hertzberg, J. 2008. Fusing DL reasoning with HTN planning. In *Advances in Artificial Intelligence*, volume 5243 of *Lecture Notes in Computer Science*, pp 62–69.

Kaelbling, L., and Lozano-Perez, T. 2011. Hierarchical task and motion planning in the now. In *International Conference on Robotics and Automation (ICRA)*, pp 1470 –1477.

Kang, D.; Seo, E.; Kim, S.; and Choi, H.-J. 2009. Automatically learning robot domain ontology from collective knowledge for home service robots. In *International Conference on Advanced Communication Technology (ICACT*, pp 1766–1771.

Keller, T.; Eyerich, P.; and Nebel, B. 2010. Task planning for an autonomous service robot. volume 6359 of *Lecture Notes in Computer Science*, pp 358–365.

Kemke, C. 2006. Action representation for natural language interfaces to agent systems. In *International Conference on Hybrid Information Technology (ICHIT*.

Kunze, L.; Roehm, T.; and Beetz, M. 2011. Towards semantic robot description languages. In *International Conference on Robotics and Automation (ICRA)*.

Lortal, G.; Dhouib, S.; and Gérard, S. 2011. Integrating ontological domain knowledge into a robotic dsl. In *International Conference on Models in Software Engineering (MODELS)*, pp 401–414.

Nau, D.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research* 20:pp 379–404.

Prud'hommeaux, E., and Seaborne, A. 2008. *SPARQL Query Language for RDF*. W3C. http://www.w3.org/TR/rdf-sparql-query/.

Shilane, P.; Min, P.; Kazhdan, M.; and Funkhouser, T. 2004. The princeton shape benchmark. In *In Shape Modeling International*, pp 167–178.

Sirin, E.; Parsia, B.; Wu, D.; Hendler, J.; and Nau, D. 2004. Htn planning for web service composition using shop2. *Web Semantics: Science, Services and Agents on the World Wide Web* 1(4):pp 377–396.

Sohrabi, S.; Baier, J. A.; and McIlraith, S. A. 2009. Htn planning with preferences. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp 1790–1797.

Tenorth, M., and Beetz, M. 2009. KnowRob - Knowledge Processing for Autonomous Personal Robots. In *International Conference on Intelligent Robots and Systems (IROS)*, pp 4261–4266.

Tenorth, M.; Perzylo, A. C.; Lafrenz, R.; and Beetz, M. 2012. The RoboEarth language: Representing and Exchanging Knowledge about Actions, Objects, and Environments. In *International Conference on Robotics and Automation (ICRA)*. Accepted for publication.

Tsuneto, R.; Nau, J. H. D.; Hendler, J.; Nau, D.; and Barros, L. N. D. 2007. Matching problem features with task selection for better performance in htn planning. In *AIPS Worksop on Knowledge Engineering and Acquisition for Planning*.

Waibel, M.; Beetz, M.; Civera, J.; D'Andrea, R.; Elfring, J.; Galvez-Lopez, D.; Häussermann, K.; Janssen, R.; Montiel, J.; Perzylo, A.; Schiessle, B.; Tenorth, M.; Zweigle, O.; and van de Molengraft, R. 2011. Roboearth: A world wide web for robots. *IEEE Robotics Automation Magazine (RAM)* 18:pp 69–82.

Warfield, I.; Hogg, C.; Lee-Urban, S.; and Muñoz-Avila, H. 2007. Adaptation of hierarchical task network plans. In *International Conference of the Florida AI Research Society (FLAIRS)*, pp 429–434.

Yu, L. 2007. *Introduction to the Semantic Web and Semantic Web Services*. Chapman & Hall, 1 edition.