ICAPS 2006

The English Lake District, Cumbria, UK

Workshop on Plan Analysis and Management

VSS

Lee McCluskey
University of Huddersfield, U.K.

Karen Myers
SRI International, USA

Biplav Srivastava
IBM India Research Lab, India














ICAPS 2006

The English Lake District, Cumbria, UK

Workshop on Plan Analysis and Management

Lee McCluskey
University of Huddersfield, U.K.

Karen Myers
SRI International, USA

Biplav Srivastava
IBM India Research Lab, India

WS5

ICAPS 2006
Workshop on Plan Analysis and Management

Table of contents

ICAPS 2006 Workshop on Plan Analysis and Management

Preface

Planning has primarily considered plans in the context of how they could be generated from the domain theory and problem specification. While synthesizing plans is useful and important, it puts heavy demand on how the information is modeled and available in an application. Beyond synthesis, they may also be acquired from domain meta-models or experts, or learnt.

As planning is employed in new applications like grid, web services, e-services, workflows and business processes to facilitate service composition to manage tasks and resources, there is a growing realization that acquiring plans is just one part of the complex process of how a plan is used in any real world application. Users want to generate plans only when needed and they want to know how they are using their plans. Once available, they may be organized in large collections, where they can be grouped along different purposes and are made amenable to the search, inspection, evaluation, and modification by human experts or automated reasoning systems. Eventually, plans will outlast their utility and be replaced by newer, better ones. This has lead some researchers to propose a life-cycle view of plans to bring together the various techniques needed to use plans in an application.

In this workshop, we will look at issues related to how plans are used after they are obtained and how they play a role in subsequent planning. For our purpose, we will consider plans and workflows as synonymous to stress the fact that beyond the PDDL plan representation, planning should also consider the generalized workflows representation which is not always automatically generated. The workshop will look at issues of acquiring plans (e.g., eliciting plans, learning them), synthesizing plans when the domain is incomplete, modeling updates to plans (e.g., through instructions), analysis techniques to manage plans, metadata generation, storage of plans and how to retire them. The workshop will bring together researchers who are working on these topics to discuss these issues as well as real systems that are under development.

Lee McCluskey, Karen Myers, Biplav Srivastava

Organizers

- *Lee McCluskey, t.l.mccluskey@hud.ac.uk, University of Huddersfield, U.K.*
- *Karen Myers, myers@ai.sri.com, SRI International, USA*
- *Biplav Srivastava, sbiplav@in.ibm.com, IBM India Research Lab, India*

Programme Committee

- *Jim Blythe, blythe@isi.edu, USC Information Sciences Institute, USA*
- *Subbarao Kambhampati, rao@asu.edu, Arizona State University, Tempe AZ, USA.*

Recording Rationale in <I-N-C-A> for Plan Analysis

Gerhard Wickler, Stephen Potter, Austin Tate

Artificial Intelligence Applications Institute
University of Edinburgh, Edinburgh, Scotland
{g.wickler, s.potter, a.tate}@ed.ac.uk

Abstract

The aim of this paper is to show how the rationale behind a plan can be recorded in the plan itself. The <I-N-C-A> model which underlies the I-X framework will be described in detail, focussing on annotations. It is there that a planner can record the justifications for including components into the plan. Recording rationale information of this type can be used for a number of purposes in the life cycle of a plan, including plan indexing and retrieval, failure recovery, plan explanation and establishing trust as explained in this paper.

Introduction

Plans are the artefact that is the result of the planning process. Traditionally, a plan is described as a set of activities together with some organizational structure, e.g. a sequence in the simplest case (Ghallab *et al*., 2004). This is a simplistic model of a plan that can only be applied in toy domains where plans cannot go wrong, need not be stored for later re-use, need not be justified, etc. This view of plans ignores a lot of the knowledge that is generated and used during the planning process. In this paper we will describe the <I-N-C-A> model of a plan (Tate, 2003) which can store annotations to record knowledge about the plan that is generated during the planning process. Specifically, we want to record the rationale behind some (but not all) planning decisions in the plan itself for later use (e.g. during plan execution, re-planning or explanation generation). This knowledge can be used to facilitate plan analysis and help maintain the plan as a meaningful entity.

Background

Rationale has been recognized as an important type of information in the planning literature. In fact it can be traced back to early work on Hacker's plan teleology (Sussman, 1973), Nonlin's "Goal Structure" (Tate, 1977; Tate, 1983) and work on Plan Rationale in SIPE (Wilkins, 1988). Plan rationale capture and use is a key research objective in the I-X framework (Tate, 2000; Potter *et al*., 2003; Wickler *et al*., 2006) and its predecessor O-Plan (Currie and Tate, 1991; Tate *et al*., 2000).

One of the fundamental ideas here is that it is necessary to have a clear and sharable ontology of plans before one can reason about plans (Tate, 1996; 1998). The <I-N-C-A> model of a plan represents such an ontology and will be described in detail in this paper, focusing on the component that is used to record the rationale behind the plan.

Rationale is an essential component of knowledge-rich plans (Polyak and Tate, 1998). Having such models not only facilitates the planning process itself, but also makes it possible to analyse and re-use such plans. Plan rationale can be viewed in terms of causality, dependencies and decisions. Each of these dimensions addresses practical issues in the planning process and adds value to the resultant plan.

The <I-N-C-A> Model in I-X

<I-N-C-A> is a generic model for synthesis tasks (Tate, 2003). While its level of abstraction makes it possible to apply the generic model to a wide variety of tasks, it assumes a more specific meaning in the I-X agent framework when the object to be synthesized is a plan, a course of action the I-X agent intends to follow.

Terminology

In this section we will introduce some of the terminology used in the description of <I-N-C-A> that follows. This is necessary as we use the terms explained here with specific meanings.

World-State Propositions. We assume here that a state of the world can be described by a set of world-state propositions. By a world-state proposition we mean any logical expression that represents a proposition about the world that can be true or false, and not necessarily a proposition in propositional logic. <I-N-C-A> does not commit to any specific formalism for world-state propositions. Traditionally world-state propositions are described as first-order literals or state-variable expressions in AI planning, but more complex formalisms may be required to reason about, for example, the knowledge of agents in a world state.

Primitive and Complex Activities. Primitive activities are considered to be the atomic elements that make up the plan. They are primitive in the sense that, from the perspective of the planner, they can be executed directly.

A primitive activity must be an instantiation of some activity schema defined in the planning domain. An activity schema contains variables representing the parameters necessary to describe fully the activity: For primitive activities to be executable these parameters must

have specific values. The name of each activity schema must be unique within a planning domain, whereas there can be multiple primitive activities with the same activity name in a plan. In classical planning primitive activities are often called actions (Ghallab *et al*., 2004).

Complex activities are not primitive in that, from the perspective of the planner, they cannot be executed directly but instead need to be refined or broken down into primitive activities that can be executed. In Hierarchical Task Network (HTN) planning complex activities are often called tasks (Ghallab *et al*., 2004) or processes (Tate, 1998). Together, primitive and complex activities constitute the set of all activities. Note that this terminology applies at the object-level, i.e., referring to entities in the domain in question, as well as at the meta-level relating to the planning process itself, as described below.

Note too that the choice of which activities are primitive and which complex depends on the context and knowledge of the agent in question: usually an activity will be modelled as primitive if it can be carried out in one step from this perspective, and as complex otherwise.

Plans. An instantiation of the <I-N-C-A> model is an <I-N-C-A> object. In the I-X framework an <I-N-C-A> object is synonymous with a plan. A plan can be partial in the sense that it is not (yet) an actionable solution to a planning problem. It is the job of the planner to refine a partial plan into a solution plan.

The <I-N-C-A> Representation in I-X

Planning can be described as synthesizing an <I-N-C-A> object, i.e., a plan, in which *nodes* are activities. We can formally define an <I-N-C-A> object in I-X as a 4-tuple (*I,N,C,A*) consisting of:

- a set of *issues I*,
- a set of *activity nodes N*,
- a set of *constraints C*, and
- a set of *annotations A*.

Issues. *I* is the set of unresolved *issues* in the current plan, i.e., in this <I-N-C-A> object. An issue is represented by a syntactic expression of the form l:$M(O_1,\ldots,O_n)$, where:

- l is a unique label for this issue,
- M is a symbol denoting a primitive plan modification activity, and
- $O_1,\ldots,O_n$ are plan-space objects, i.e. they are issues, nodes, constraints or annotations. The number of such objects, n, and the interpretation of each object in the context of the issue, will depend on the particular primitive plan modification activity represented by this issue.

Issues can be seen as primitive meta-level activities, i.e. things that need to be done to the plan before it becomes a solution to a given planning problem. This approach is inherited from O-Plan (Currie and Tate, 1991; Tate *et al*., 2000) and is also seen in planners such as OPIS (Smith, 1994). The most commonly found primitive meta-level activities carried out by planners, but usually only implicit in their underlying implementation or internal plan representation, are:

- Achieving a goal (in classical planners): Let p be a world-state proposition and τ be a time point, then the primitive meta-level activity of achieving p at τ can be represented as the issue:

$$l_1\texttt{:achieve}(p,\tau)$$

- Accomplishing a complex activity (in HTN planners): Let $a \in N$ be a complex activity. Then the primitive meta-level activity of accomplishing a can be represented as the issue:

$$l_2\texttt{:refine}(a)$$

Here, `achieve` and `refine` are examples of symbols denoting primitive plan modification activities. Note that these symbols are not domain specific but specific to the planning process by which these types of issue are handled. Issues can be either 'negative', in which case they can be thought of as flaws in the plan, or they can be 'positive', e.g., opportunities.

An alternative view of issues now being explored in recent I-X research is to see them as always expressed as questions that need to be answered. For example, the primitive meta-level activity of refining a can be phrased as the question "How can a be accomplished?" Adopting this view, issues can then be classified and manipulated according to the question types (Conklin, 2005) described in recent advances based on the large body of work on issue-based design (Conklin and Begeman, 1988).

An <I-N-C-A> object is considered to be a solution to a planning problem only if the set of issues is empty.

Nodes. N is the set of activities (*nodes*) to be performed in the current plan, i.e., in this <I-N-C-A> object. An activity is a syntactic expression of the form l:$\alpha(o_1,\ldots,o_n)$, where:

- l is a unique label for this activity,
- α is a symbol denoting an activity name, and
- $o_1,\ldots,o_n$ are object-level terms, i.e. they are either constant symbols describing objects in the domain, or they are as yet uninstantiated variables standing for such objects.

Time points constitute a special class of domain objects that are found as parameters of an activity. Specifically, two time points, one representing the begin and the other the end of an activity, are often used as parameters.

In the context of I-X, nodes represent the object-level activities in the plan, i.e., things that need to be performed by some agent to execute the plan. As mentioned above, activities can be of two types from the perspective of the planner:

- Primitive activities: primitive activities can be carried out directly by an agent executing the plan. For example, in a search and rescue domain, the primitive activity of flying the aircraft `ac1` from location `loc1` to location `loc2` may be represented as:

$$l_3\texttt{:fly(ac1,loc1,loc2)}$$

- Complex activities: complex activities cannot be accomplished directly by the agent executing the plan but need to be refined into primitive activities. For example, the complex activity of rescuing an isolated person `ip` may be represented as:

$$l_4\texttt{:rescue(ip)}$$

In this example, `fly` is a primitive activity symbol and `rescue` is a complex activity symbol in some domain. Activity symbols have to be domain specific. It follows that there has to be an activity schema defined for the domain that has the name `fly` and describes when this activity schema is applicable and how it will change the world when applied, and there has to be a refinement defined in the domain that accomplishes a complex activity with the name `rescue` and describes how exactly it can be accomplished.

Note that the set N of activities in the plan may contain both complex activities and the primitive activities that have been chosen to implement them.

Constraints. C is the set of *constraints* that must be satisfied by the current plan (<I-N-C-A> object). A constraint is a syntactic expression of the form l:$c(v_1,\ldots,v_n)$, where:

- l is a unique label for this constraint,
- c is a symbol denoting a constraint relation, and
- $v_1,\ldots,v_n$ are constraint variables, i.e., they can represent domain objects (including time points), variables in activities (which may have binding constraints attached).

Constraints represent the relations that must hold between the different objects related in the constraints for the plan to be executable. In the context of planning, the most commonly used constraints are of the following types:

- Ordering constraints: Let v_1, v_2 be variables in the plan representing time points. Then the constraint that v_1 has to be before v_2 can be represented as:

$$l_5\texttt{:before}(v_1,v_2)$$

- World-state constraints: Let p be a world-state proposition and v a variable representing a time point in the plan. Then the fact that p is a condition that has to hold at the time point represented by v, or the fact that p is an effect of an activity that holds at time point v can be represented respectively as:

$$l_6\texttt{:cond}(p,v)$$

$$l_7\texttt{:effect}(p,v)$$

- Variable binding constraints: Let v be a variable mentioned in some activity $a\in N$ and s be a constant symbol in the planning domain. Then the fact that v must take the value s can be represented as:

$$l_8\texttt{:value}(v,s)$$

These are just some of the constraint types that can be defined. The objects related to each other can be of different types. This is reflected by the domains of the constraint variables representing them. They can be world-state propositions as in conditions and effects, or they can be variables used in activities representing time points or other domain objects in the plan as in ordering and variable binding constraints.

Annotations. A is the set of *annotations* attached to the current plan. Amongst other things, annotations can be used to add human-centric information to the plan. They may be informal or they may adhere to some detailed syntax (which is not specified as part of <I-N-C-A>).

Annotations can be used to record arbitrary information about the plan (and the annotations form a part of this plan – hence the plan becomes, in some sense, self-descriptive). Specifically, in this paper we want to discuss the annotation of plans with one particular type of rationale, namely the rationale information that can be recorded by the planner during the planning process. In this case, an annotation will be a syntactic expression of the form l_a:r (l_p:O, l_m:M, $O_1,\ldots,O_n$) , where:

- l_a: is a unique label for this annotation,
- r is a rationale predicate relating a plan-space object to other plan-space objects,
- l_p:O is a labelled plan-space object that is part of the current plan, i.e., it is an issue, an activity, a constraint or an annotation,
- l_m:M is an issue that was formerly in the plan and has since been resolved, i.e., it is a primitive meta-level activity that has been performed by the planner, and
- $O_1,\ldots,O_n$ are plan-space objects that may or may not be labelled.

An annotation of this type represents the fact that the plan-space object O was introduced into the plan as part of performing the plan modification activity M, and possibly involving other plan-space objects $O_1,\ldots,O_n$. The rationale predicate r denotes the relationship between these objects and describes the justification for including O. Thus, the interpretation of such an annotation depends on the rationale predicate r used. The different labels are necessary to specify the exact object that is being referred to. This is necessary as there might be two activities in the plan which are identical except for the label. The following examples illustrate the use of rationale annotations of this form.

- Let l_m:`achieve`(p,τ) be an issue in the current plan and let $\alpha(o_1,\ldots,o_n)$ be an activity schema defined in the domain that has an effect that unifies with p under the substitution σ. Suppose the planner introduces a new activity l_p:$\sigma(\alpha(o_1,\ldots,o_n))$ into the plan to address the issue l_m:`achieve`(p,τ). Then the following annotation can be added to the plan to record the rationale for adding l_p:$\sigma(\alpha(o_1,\ldots,o_n))$:

  `naap`$(l_p$:$\sigma(\alpha(o_1,\ldots,o_n)),l_m$:`achieve`$(p,\tau),p)$

  In this case `naap` is a rationale predicate that expresses that a new activity, the first argument, was introduced into the plan to address the issue of achieving some proposition (the second and third arguments respectively). Thus, the argument types for this particular rationale predicate are an activity $a\in N$, an issue $M\in I$ in which the plan modification activity symbol is

`achieve`, and a world-state proposition. Furthermore, the last argument, the proposition p, must be the same as the one to be achieved in the plan modification activity, and it must be unifiable with one of the effects of the activity $a \in N$.

In this case, a second rationale annotation could be introduced in a similar fashion to express the fact that l_p:$\sigma(\alpha(o_1,\ldots,o_n)$ has to be performed before the time point τ.

- Let l_m:`refine(`a`)` be an issue in the current plan and let there be a refinement Δ defined in the domain that can be used to accomplish a under the substitution σ by refining it into, amongst other things, activities $\sigma(\alpha_1(o_1,\ldots,o_n))\ldots\sigma(\alpha_k(o_1,\ldots,o_n))$. Note that the elements into which a is refined can together be seen as an <I-N-C-A> object, i.e. they can be issues, nodes, constraints and annotations. Suppose the planner uses Δ to refine a and this adds new activities l_{p1}:$\sigma(\alpha_1(o_1,\ldots,o_n))\ldots l_{pk}$:$\sigma(\alpha_k(o_1,\ldots,o_n))$ to N to address the issue l_m:`refine(`a`)`. Then, the following annotation can be added to the plan to record the rationale for adding each l_{pi}:$\sigma(\alpha_i(o_1,\ldots,o_n))$, $1 \le i \le k$:

  `nadi(`l_{pi}:$\sigma(\alpha_i(o_1,\ldots,o_n))$`,`$l_m$:`refine(`$a$`),`$\Delta$`)`

  (One such annotation would be added for each new activity α_i.) In this case `nadi` is a rationale predicate that expresses that a new activity, the first argument, was introduced into the plan to address the issue of refining some proposition in accordance with some particular refinement in the domain (the second and third arguments respectively). Thus, the argument types for this rationale predicate must be an activity $a \in N$, an issue $m \in I$, where the plan modification activity symbol has to be `refine`, and a refinement. Furthermore, the last argument, the refinement Δ, must be defined as accomplishing a complex activity that can be unified with a.

  Similarly, if appropriate, analogous rationale annotations could be introduced to express the fact that other <I-N-C-A> elements of the refinement – such as issues or constraints – were also introduced as part of this refinement.

Rationale predicates of this type are usually specific to a type of issue. Hence, `naap` rationale will always relate to an `achieve` issue, and `nadi` rationale will always relate to a `refine` issue. However, there may be multiple rationale predicates that may be used with the same issue – that used will depend on how the planner did actually resolve the issue. For example, achieving a proposition at some time point can be done by introducing a new activity before the time point or by maintaining the truth of the proposition if it was true at another, previous time point. Thus, the relation between rationale predicates and issues is not one-to-one: issues need not always be resolved in the same manner.

Note too that this type of rationale, recording justifications for the inclusion of objects into the plan, is only one type of rationale that one may want to record in a plan. For example, we may want to record why a specific way of refining a plan was chosen among the various available options. While we believe that this type of information would be very useful to record, we believe that this will best be approached by use of a separate decision structure. It is in general not possible to extract useful knowledge of this kind from a search-based planning algorithm that tries out many possibilities and backtracks upon failure. At any choice point, there may be a large number of reasons why all the leaf nodes that are in the search space under the choice point represent failures in the search, and it may be hard to abstract these into meaningful rationale. However, there also exist choice points in a search space where a decision is forced or made via user selection from open alternatives and it may be most useful to record this as part of the rationale for the plan. This is not described here though.

Issues as Questions

In the I-X framework, until recently, issues had a task or activity orientation to them, being mostly concerned with actionable items referring to the process underway – i.e., actions in the process space. This is now not felt to be appropriate, and we are adopting the gIBIS (Conklin and Begeman, 1988) orientation of expressing these issues as any of a number of specific types of question to be considered (Selvin, 1999; Conklin, 2005). The types of questions advocated are:

1. Deontic questions – What should we do?
2. Instrumental questions – How should we do it?
3. Criterial questions – What are the criteria?
4. Meaning or conceptual questions – What does *X* mean?
5. Factual questions – What is *X*? or Is *X* true?
6. Background questions – What is the background to this project?
7. Stakeholder questions – Who are the stakeholders of this project?
8. Miscellaneous questions – To act as a catch all.

The first 5 of these are likely to be the most common in our task support environment. This is similar to the Questions - Options - Criteria approach (MacLean *et al*., 1991) - itself used for rationale capture for plans and plan schema libraries in our earlier work (Polyak and Tate, 1998; 2000) and similar to the concept mapping approaches used in Compendium (Selvin et al. 2001). Compendium can in fact exchange its set of issues, activities and some types of constraints and annotations with I-X (Buckingham Shum *et al*., 2002; Chen-Burger and Tate, 2003).

The Uses of Rationale

Fundamental to the <I-N-C-A> model is the idea of maintaining annotations as first-class elements placed alongside the more conventional elements of a plan. One of the principal uses of annotations is to capture rationale; hence, we consider rationale to be an important element of

this model, and rationale capture and expression are areas which we are currently exploring.

The approach outlined in the previous sections, , should be seen as a framework and tentative steps towards defining a typology of plan rationale and corresponding mechanisms for its capture. These tasks are necessarily guided by the uses to which we want to put this rationale; hence, in this section we discuss briefly some of the types of operations and reasoning that we hope to support through the capture of rationale. In general terms, these are intended to support activity in *real* domains (as opposed to classical planning domains and puzzles). In other words, domains in which we accept that information and knowledge may be imprecise, incorrect or missing, and as a result, we expect plans to fail – and expect that the use of rationale will enable us to *fail better*.

Explanation and Trust

As might be expected, a major use of rationale is for explaining the existence of particular elements in the plan, e.g., why a certain activity (rather than any other) appears in the plan. This becomes particularly important when trying to decide if the plan can be re-applied in the current context, or if execution of the plan fails or partially fails (of which, more later). Another use of explanation, one particularly important in mixed-initiative (i.e., human and computer) agent systems arises when we wish to *justify* a certain activity, particularly in those cases where we are asking another agent to perform this activity. In all but the most rigidly enforced hierarchical systems, where agents simply obey commands (and which occur very rarely in practice), we should expect that any agent might respond to such a request with a request of its own demanding that the activity be justified (and that, if the activity cannot be justified to the agent's satisfaction, it might refuse to perform the activity). It should be apparent that rationale would allow us to supply some justification. Moreover, through the use of <I-N-C-A> objects as our common interlingua in the domain, this justification can be included and communicated as part of the activity. In this way, the object may be thought of as analogous to the idea of proof-carrying code, in that the presence of the rationale can help convince the recipient of the appropriateness of performing the activity and that it is 'safe' to be performed in the current situation.

This sort of transaction and reasoning can be seen as an important step for establishing *trust* between agents. Notions of trust, and ways in which it can be established and managed, are currently receiving much attention among those considering open agent architectures, particularly Semantic Web and Semantic Web Services researchers, where it is considered to be vital if these initiatives are to come to full fruition.

Plan Indexing and Retrieval

Often re-use of existing plans will be more appealing than planning anew for a particular task. One use of rationale is for richer indexing (and later retrieval) of plans; alongside the description of what the plan does (expressed in the through the plan itself), and the constraints under which it is applicable. The rationale annotations allow us to access the reasons why the plan does what it purports to do. Properly captured, this information would allow us to avoid plan re-use under inappropriate conditions or avoid choosing plans that are based on (what are now known to be faulty) assumptions or judgements, or at least to be aware of these limitations and deal with them accordingly.

Failure Recovery and Replanning

In the real world it is inevitable that some plans will fail; even the best-laid plans can be undermined by some unexpected event. The failure may or may not be important with respect to the plan rationale. We need to separate unimportant minor side-effects from failures which impact on the intended results of the plan (Tate, 1984; Reece and Tate, 1994; Drabble and Tate, 1997). In such cases, it is very likely that we will need to do something to recover, and to do this efficiently, we will need to try to understand why the plan has failed, and hence, when replanning to help guide the choice of alternative actions that may overcome this failure.

Explanation-Based Plan Learning

Since the plan is accompanied by some explanation of why it is considered valid (in the form of the rationale), this suggests the possibility of learning about the domain from both positive and negative examples (plan successes and failures). This learning may help to, for instance, identify and repair faulty knowledge or assumptions, and provide modified rational criteria for choice of particular options over others.

Conclusions

In this paper we have presented an approach to recording the rationale behind a plan in the plan itself, thus making the plan a self-contained entity that does not require knowledge of the planning algorithm to explain the structure of the plan. Fundamental to this approach is the <I-N-C-A> model which can be used to describe synthesis tasks and has been used in the I-X framework for synthesizing plans. Issues in this model can be described as meta-level activities that are performed by the planner to refine the plan. During this planning process the planner adds new constraints on the space of possible behaviour to the plan, and each of these constraints is added for a reason. It is this type of rationale that we can record as annotations in <I-N-C-A> in order to be able to better understand the plan, the result of the planning process. This knowledge-rich plan can then be used in various ways outlined in this paper, thus facilitating the use of the plan in a wider context.

Acknowledgments

The I-X project is sponsored by the Defense Advanced Research Projects Agency (DARPA) under agreement number F30602-03-2-0014. Parts of this work are supported by the Advanced Knowledge Technologies (AKT) Interdisciplinary Research Collaboration (IRC) sponsored by the UK Engineering and Physical Sciences Research Council by grant no. GR/N15764/01. The University of Edinburgh and research sponsors are authorized to reproduce and distribute reprints and on-line copies for their purposes notwithstanding any copyright annotation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of other parties.

References

Buckingham Shum, S., De Roure, D., Eisenstadt, M., Shadbolt, N. and Tate, A. (2002) CoAKTinG: Collaborative Advanced Knowledge Technologies in the Grid, *Proceedings of the Second Workshop on Advanced Collaborative Environments*, Eleventh IEEE Int. Symposium on High Performance Distributed Computing (HPDC-11), July 24-26, 2002, Edinburgh, Scotland.

Chen-Burger, Y. and Tate, A. (2003) Concept Mapping Between Compendium and I-X, Informatics Report Series, University of Edinburgh, EDI-INF-RR-0166, May 2003.

Conklin J. and Begeman M. L. (1988). gIBIS: A hypertext tool for exploratory policy discussion. *ACM Transactions on Office Information Systems*, 4(6), pp 303-331.

Conklin J. (2005) Dialogue Mapping: Building Shared Understanding of Wicked Problems, Wiley.

Currie K. and Tate A.. (1991) O-Plan: the Open Planning Architecture. *Artificial Intelligence*, Vol. 52, pp 49-86.

Drabble, B. and Tate, A. (1997) Repairing Plans on the Fly, in *Proceedings of the NASA Workshop on Planning and Scheduling for Space*, Oxnard CA, USA, October 1997

Ghallab M., Nau D., and Traverso P. (2004) *Automated Planning – Theory and Practice*,. Elsevier/Morgan Kaufmann.

MacLean A., Young R., Bellotti V. and Moran T. (1991) Design space analysis: Bridging from theory to practice via design rationale. In *Proceedings of Esprit* '91, pages 720-730, Brussels, November 1991.

Polyak S. and Tate A. (1998) Rationale in Planning: Causality, Dependencies and Decisions, *Knowledge Engineering Review*, Vol.13 (3), pp. 247-262, September, 1998, Cambridge University Press. See http://www.aiai.ed.ac.uk/project/oplan/documents/1998/98-rationale.pdf

Polyak S. and Tate A. (2000) A Common Process Ontology for Process-Centred Organisations, Knowledge based Systems, 2000. Earlier version by S. Polyak published as University of Edinburgh Department of Artificial Intelligence Research paper 930, 1998. See http://www.aiai.ed.ac.uk/project/oplan/documents/1999/99-sebpc-cpm.pdf

Potter, S., Tate, A. and Dalton, J. (2003) I-X: Task Support on the Semantic Web, Poster Abstract, Second International Semantic Web Conference (ISWC-2003), Sanibel Island, Florida, October 2003.

Reece, G. and Tate, A. (1994) Synthesizing Protection Monitors from Causal Structure, in *Proceedings of the Second International Conference on Planning Systems (AIPS-94)*, Chicago, June 1994, AAAI Press.

Selvin A.M. (1999) Supporting Collaborative Analysis and Design with Hypertext Functionality, Journal of Digital Information, Volume 1 Issue 4.

Selvin, A.M, Buckingham Shum, S.J., Sierhuis, M., Conklin, J., Zimmermann, B., Palus, C., Drath, W., Horth, D., Domingue, J., Motta, E. and Li, G. (2001) Compendium: Making Meetings into Knowledge Events, *Knowledge Technologies* 2001, Austin TX, USA, March 4-7, 2001.

Smith, S.F. (1994) OPIS: A methodology and architecture for reactive scheduling. In Zweben, M. and Fox, M.S. (eds), *Intelligent Scheduling*, chapter 8, pages 29--66. Morgan Kaufmann, San Francisco, CA, USA.

Sussman, G. J. (1973) "A Computational Model of Skill Acquisition," *Technical Report 297*, MIT AI Lab, Cambridge, MA.

Tate, A., (1975) "Generating Project Networks", *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)* pp. 888-893, Boston, Mass. USA, August 1977. Reprinted in *Readings in Planning,* Morgan-Kaufmann, 1990.

Tate A. (1983) The Less Obvious Side of Nonlin. Department of Artificial Intelligence, University of Edinburgh. http://www.aiai.ed.ac.uk/project/oplan/documents/1990-PRE/1983-unpub-tate-nonlin-less-obvious.pdf

Tate, A. (1984) Planning and Condition Monitoring in a FMS, *Proceedings of the International Conference on Flexible Automation Systems*, Institute of Electrical Engineers, London, UK, July 1984.

Tate A.. (1996) Towards a Plan Ontology. *AI*IA Notiziqe* (Quarterly Publication of the Associazione Italiana per l'Intelligenza Artificiale), Special Issue on "Aspects of Planning Research", Vol. 9. No. 1, pp.19-26 - March 1996

Tate A.. (1998) Roots of SPAR - Shared Planning and Activity Representation, *The Knowledge Engineering Review*, Vol 13(1), pp. 121-128, Special Issue on Putting Ontologies to Use (eds. Uschold. M. and Tate, A.), Cambridge University Press, March 1998.

Tate, A. (2000) Intelligible AI Planning, in Research and Development in Intelligent Systems XVII, *Proceedings of ES2000, The Twentieth British Computer Society Special*

Group on Expert Systems International Conference on Knowledge Based Systems and Applied Artificial Intelligence, pp. 3-16, Cambridge, UK, December 2000, Springer.

Tate A.., Dalton J. and Levine J. (2000) O-Plan: a Web-based AI Planning Agent, AAAI-2000 Intelligent Systems Demonstrator, in *Proceedings of the National Conference of the American Association of Artificial Intelligence* (AAAI-2000), Austin, Texas, USA, August 2000.

Tate A.. (2003) An Ontology for Mixed-initiative Synthesis Tasks. *Proceedings of the Workshop on Mixed-Initiative Intelligent Systems (MIIS)* at the International Joint Conference on Artificial Intelligence (IJCAI-03), Acapulco, Mexico, August 2003, pp 125-130.

Wickler G., Tate A., and Potter S. (2006) Using the <I-N-C-A> Artifact Model as a Shared Representation of Intentions for Emergency Response Coordination. In: Jennings N., Tambe M., Ishida T., Ramchurn G. *Proc. First International Workshop on Agent Technology for Disaster Management*, Japan, May 2006.

Wilkins, D.E. (1988) Practical Planning: Extending the Classical AI Planning Paradigm, Morgan Kaufmann, San Mateo, CA, USA.

Tool Support for Planning and Plan Analysis within Domains embodying Continuous Change

T. L. McCluskey and R. M.Simpson
School of Computing and Engineering
The University of Huddersfield, Huddersfield HD1 3DH, UK
t.l.mccluskey,r.m.simpson@hud.ac.uk

Abstract

In this paper we describe work in progress that aims to develop a domain-independent tool set which supports the creation and analysis of domain descriptions and plans containing continuously changing processes, instantaneous events, and actions. The tools described are (i) a life history editor that enables a designer to create a domain description diagrammatically, automatically generating much of the textual encoding; (ii) an animator that simulates plans, firing events and processes; (iii) an HTN planner that generates plans in such domains. Tools (i) and (ii) are currently implemented as a subset of the GIPO III environment. We conclude by stating some initial findings and points that might be raised at the workshop.

Introduction

Planning in domains which are represented with rich notations has long been a great challenge for AI (Bresina *et al.* 2002). In particular, many real world phenomena are outside the explicit control of plan execution, yet have to be reasoned with during plan generation. Changes ocurring because of fuel consumption, continuous movement, or environmental conditions may not be adequately modelled through instantaneous or even durative actions; rather these require modelling as continuously changing processes.

Our research is heavily involved with knowledge formulation of domains which involve AI Planning. Even small, classical planning domain descriptions are difficult to encode adequately and correctly. Without tool support, encoding descriptions require the user to understand the planner and the description language in great detail. GIPO (McCluskey, Liu, & Simpson 2003) is a tool which supports the creation of domain descriptions, and the exploration of planning within the domain, using an interface that hides much of the technical details. Users can build a domain description using a diagrammatic interface, explore plan creation using a plan stepper, interface their domain to an external planner, and analyse the resulting plans using an animator. The generality of the graphical tools are enabled by the use of the 'object' as the common semantic thread.

Recently Long and Fox re-visited the issues of planning within domains with events and continuous processes, with the introduction of PDDL+ level 5 (Fox & Long 2001). In this paper we discuss extensions to the GIPO architecture and related tools for domain formulation, plan generation and plan analysis in such rich domains. We have adopted an underlying formalism similar to PDDL+ called *OCLplus*, but one which also supports the use of the GIPO knowledge acquisition tool.

Using *OCLplus*: Planning Domains with Processes and Events

OCLplus is derived from GIPO's object-centred language which is documented in the literature (McCluskey, Liu, & Simpson 2003). This language imposes a structure on the domain in terms of objects and object classes, and structures action representations in terms of object state transitions. Additionally in *OCLplus*, time is modelled explicitly as a real quantity. This enables simulation or planning with the model to adopt an arbitrary approximation to the advancement of time; hence in the domain description time is modelled as a continuous real variable, but in simulation an approximation has to be adopted. This presents a set of serious problems as discussed in the literature (Howey, Long, & Fox 2004). For example, during discrete simulation events may fire or not fire depending on the chosen granularity of time.

State changing operations are divided into three: actions, events and processes. Actions (with Strips-related semantics) bring about instantaneous change to the state of domain objects and may also update the numeric properties of those objects. Actions are the entities that agents must initiate to achieve planning goals, trigger events and processes. Events are specified in the same way as actions, but their semantics differ: whenever their preconditions are made true, the event fires and may change the state of the object(s) involved. In other words, events are automatically triggered as a result of the numeric changes brought about by domain processes, and (possibly) by state changes of other objects. Like actions, events bring about instantaneous change, but unlike actions events are not directly triggered by the plan executive. Processes specify in addition to their start condition how numeric properties of the objects in the domain are updated with the passage of time as a result of the running processes. Hence processes update numeric properties, but do not bring about state change of the objects (in the sense of changing the truth value of relations); their descriptions are

available declaratively, but they cannot be directly activated by the agents acting in the domain, despite the fact that the intention behind performing some action may simply be to start a process running.

Review of GIPO III's Life History Editor

GIPO III's object life history editor (OLHE) is used to formulate domian descriptions. Is is reminiscent of CASE tools that support designers construct diagrammatic representations of systems. These systems are usually based around an 'object model'. The CASE tool's principal function would be to help the user construct a (partial) formal model from the diagrammatic description, thus insulating the designer from the code. Depending on the CASE tool, the code generated by the tool may be some abstract, formal specification (eg tools that generate partial Z specifications), or partial implementations within a programming language.

In the case of GIPO III, the user has to create a diagrammatic representation of the planning domain using a state - machine metaphor. This forces the designer to decompose the system into an assembly of machines, where each machine represents the behaviour of a class of objects. Each machine's nodes represent the set of mutually exclusive states that an object of the class can hold. Other attributes of the objects are modelled as 'properties'. The diagrammatic form first undergoes consistency checks, and then is translated into a domain description which can be used for plan generation purposes.

Apart from an adequate diagrammatic description, the user only need specify problem scenarios (including object instances) before a target planner can be invoked. As an example, to construct the Dockworkers - robots world used in the recent AI planning textbook (Ghallab, Nau, & Traverso 2004), one only needs to construct the diagram shown is Figure 2. From this diagram GIPO III generates the dynamics of the domain, and converts this into a planner-friendly form.

The diagrams are based on primitive machines and machine transitions. Machines represent a class of objects that share the same behaviour, nodes represent the state of objects, and arcs change the state that objects go through. In addition to changing state, nodes can have changing properties (for example, locatedness) and these occur as annotated transitions of a different colour to state changes. Actions involving more than one type of object are modelled by co-ordination arcs. To capture preconditions, we constrain objects from different machines to be in certain states or have certain properties when a transition occurs. To capture necessary or conditional changes we constrain two or more transitions of objects from different machines to occur together. For more details, the reader can consult a recent paper in which we gave an informal, translator-independent semantics to these diagrams via the techniques of algebraic specification (McCluskey & Simpson 2005).

As a further aid to construction, the user is given an extensible set of machine primitives with which to work (in Figure 2 the primitives are listed in the left hand pane). This leads to a higher level platform than constructing at the level of the node and arc. For example, the dockworkers - robots world can be constructed by selecting two instances of stacks, and two bistates, and then configuring and combining them together to produce the diagram shown in Figure 2. GIPO III then produces the domain description code from the diagram, ready for input to a planner.

algorithm PlusPlan: *OCLplus* Planner

```
1.store = {node(0,init-state,empty,init-task,init-constraints)}
2.repeat
3.        call expand-plan();
4.        call simulate events-processes();
5.until empty(store) or ∃ N ∈ store:
          solution-node(N);
6.end
procedure expand-plan()
1. repeat
2.          remove node(T,S,PP,Exp,C) from store;
3.          if not methods(start(Exp),S,C) = {}then
4.            ∀M ∈ methods(start(Exp),S,C)
5.              Exp+ = Exp with M expanded;
6.              C+ = C with M's expansion constraints;
7.              store node(T,S,PP,Exp+,C+)
8.          else if not operators(start(Exp),S,C) = {}then
9.            ∀ O ∈ operators(start(Exp),S,C)
10.             store node(T,apply(O,S),add(PP,O),rem(Exp,O),C)
11.         else write node(T,S,PP,Exp,C) to temporary store
12. until empty(store) or ∃ N ∈ store
          solution-node(N);
13. let store := temporary store;
14. end
procedure simulate events-processes()
1. temporary store := store; store = empty;
2. ∀ node(T,S,PP,Exp,C) ∈ temporary store:
3.    ∀ E ∈ Events:
4.      if preconds(E,S) = true then S := apply(E,S)
5.    ∀ P ∈ Processes:
6.      if preconds(E,S) = true then S := apply(P,S)
7.    store node(T+Delta,S,PP,Exp,C);
8. end
```

Figure 1: The PlusPlan Algorithm

GIPO III's Life History Editor for *OCLplus*

The OLHE for continuous domains uses the same philosophy as the standard OLHE discussed above. Colour coding distinguishes actions, events, transitions, processes, states etc. Events are connected to the states that change as a result of the event, and an event may be co-ordinated with objects of other classes in the same way as actions. Additionally, an event will have one or more preconditions which depend on time. The specification of these properties are initiated and changed by clicking on the event's icon.

Processes are connected up diagrammatically with states that are required as their preconditions, but expressions involving continuously varying values require the use of a expression editor. We illustrate this idea with the familiar bath - filling application shown in Figure 3. In the diagram there is one process specified (filling), annotated with a

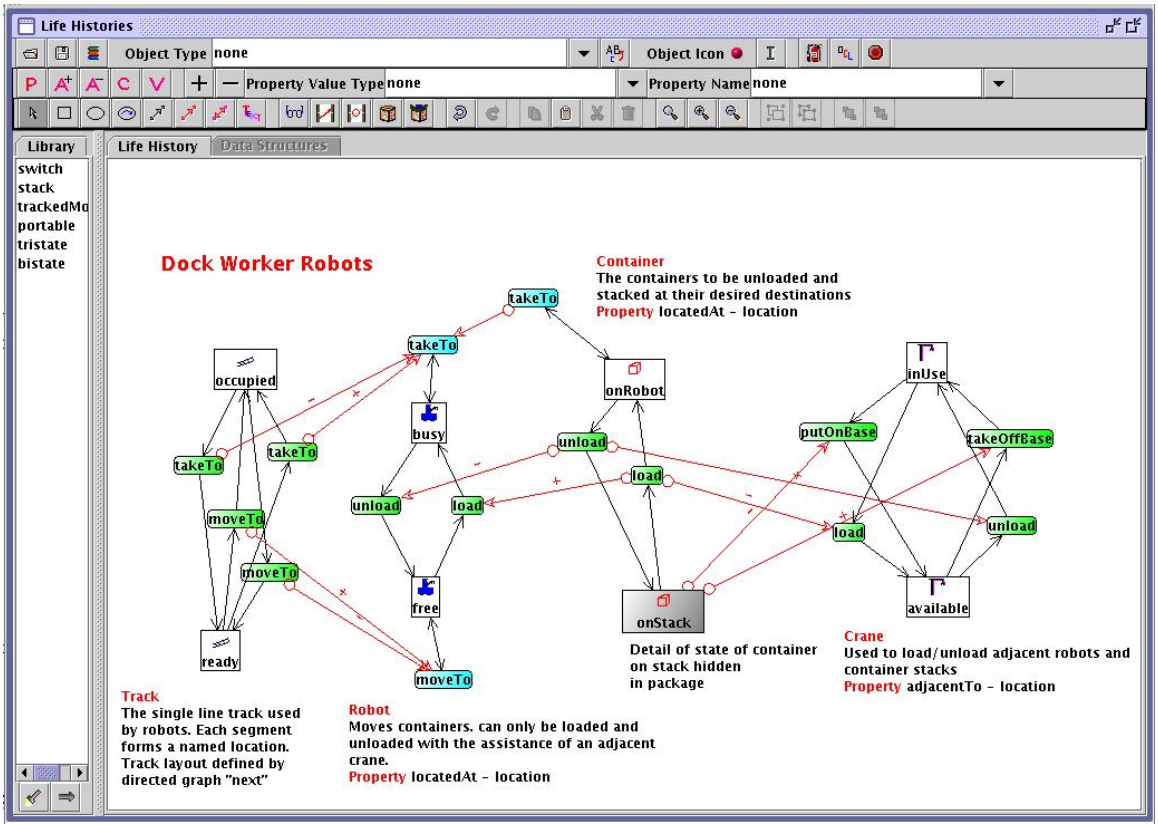

Figure 2: The Dockworkers Domain: Example of GIPO III's OLHE

clock icon to show it is a process. As well as connecting up the process to relevant states/events, the designer must input triggers for the process, i.e.

$level(Bath) \leqslant capacity(Bath)$
$flow(Bath) > 0.0$

as well as the effect of the process:

$level(Bath) = level(Bath) * flow\text{-}rate * time\text{-}increment$

An expression editor is provided for this. There is one event in the diagram - 'flood'. State change information is generated by the connections of the event in the diagram, but, like a Process, an expression editor is needed to input the fluent trigger for the event, in this case:

$level(Bath) \geqslant capacity(Bath) \& flow(Bath) > 0.0$

With this extra information, GIPO III can generate the domain description for use with the Stepper and Planner outlined below.

GIPO III's Plan Stepper

The **Plan Stepper** assumes a complete but not necessarily correct domain description exists and that the description requires dynamic testing. The stepper allows the user to manually develop a graphical representation of a plan to solve a given domain problem by choosing from the defined operators a sequence in which to apply them and by choosing instantiations of the operator parameters that legally allow the operator's application at that stage in the developing plan. We continue the simple bath filling example in Figure 4. Here the object's timelines run along from left to right, in the top pane. In the lower pane the process(es) activation is shown from left to right. Initially a plug is placed in the bath (first dot - in red) then the hot water tap is turned on (affecting the bath, the tap and having the effect of starting the filling process). Next the cold tap is turned on, and finally after several minutes the bath overflows and causes a flood (the flood event is signified by a cross).

The tool in addition to providing a graphical representation of the plan checks that each step can be legally applied assuming the given problem's initial state. In this way the user gains feedback on whether or not the domain description fulfills his/her expectation. Clearly if the plan is not legal then either the plan itself is flawed or the domain specification is at fault. The user is expected to iterate over this process until both sample plans and domain specification appear satisfactory. The plan stepper allows a user to manually create plan descriptions and have them checked against a developing formal description of the problem domain. Tools based on the idea of a plan stepper can be of more use than

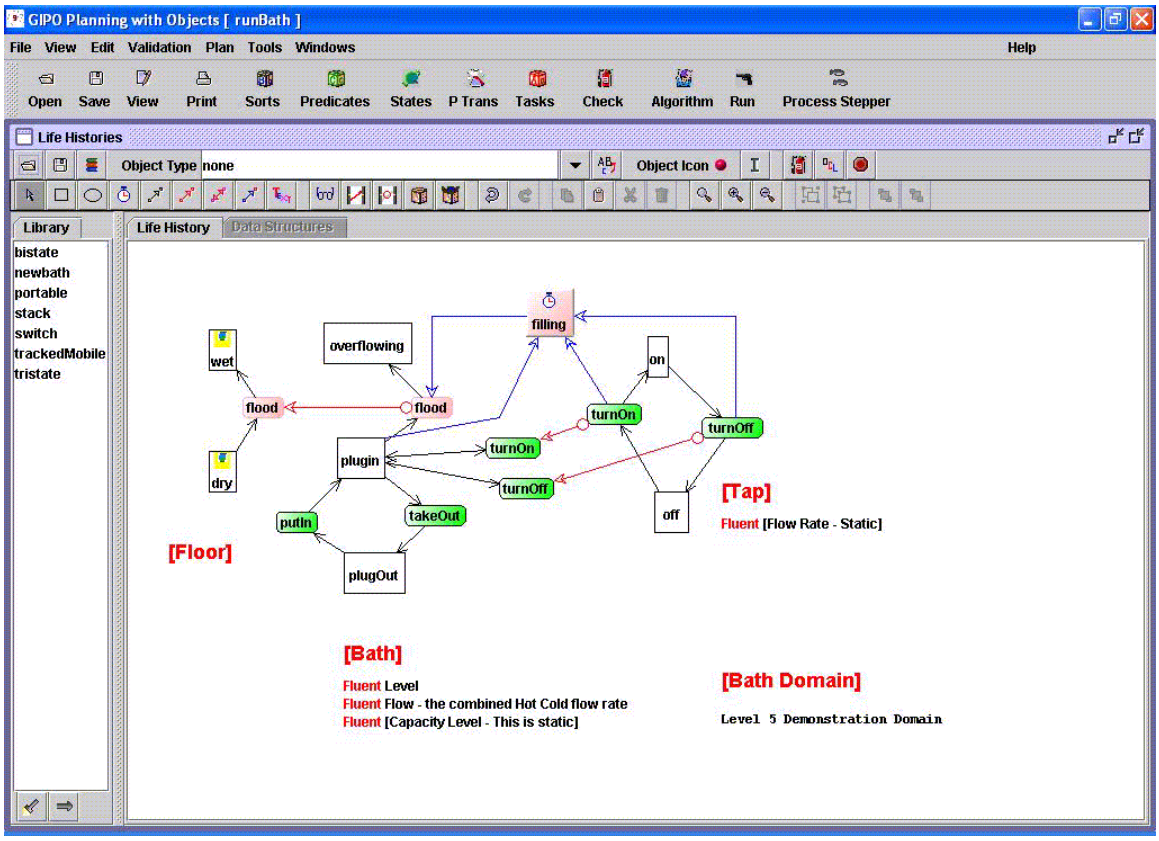

Figure 3: The Bath Filling Domain: Example of GIPO III's OLHE

just as a tool for domain description validation. In many domains users are not looking for nor would be prepared to trust fully automated tools to carry out their planning tasks.

Plan Generation

We have designed and implemented a plan generation algorithm ('PlusPlan') for *OCLplus* which was derived from HyHTN (McCluskey, Liu, & Simpson 2003). HyHTN is a hierarchical planner which, like Shop (Nau *et al.* 2000), plans by selecting and expanding method operators, and keeping an advanced state within search nodes. HyHTN's innovation is that it is equally suited to hierarchical, non-hierarchical planning, or a combination of the two. During search, nodes undergo expansion operations which replace names of methods with their decomposition. However, if within a node some condition cannot be achieved in the current state then HyHTN can perform a heuristic state space search (using the plangraph heuristic) to generate a plan to achieve the goal.

Essentially, PlusPlan works using a blend of planning and simulation. The simulation is similar to that carried out in the Plan Stepper described above: an incremental time step is used to simulate the passing of time. Planning involves a state-advancing HTN search at each instant of time. Hence, if the goal can be reached by the effect of instantaneous actions only, it will endeavour to find a plan to achieve this, and apply that plan's actions to an advancing state, in the first instant. More likely, nodes will be processed resulting in the addition of new nodes to the search, but these will not be further processed as some event or process is needed to occur or terminate before further work on a plan can occur.

Figure 1 contains an outline of PlusPlan. It searches through a space of nodes of the form:

$$node(T, S, PP, Exp, C)$$

where T is the elapsed time from the start of the plan, S is the current state at time T, PP is the plan that produces S, Exp is a set of operators representing the developing (future) plan, and C is a set of constraints on Exp. Initially Exp is the abstract task to be achieved, and C an initial set of constraints.

Procedure expand-plan tries to create and apply a plan at one instant. The resulting open nodes in 'store' are either solutions or nodes where operators cannot be executed. In line 3 $start(Exp)$ = set of fringe operators, ie those that could possibly start (temporally) Exp;
$methods(start(Exp), S, C)$ is the set of non-primitive operators in the fringe such that their preconditions are met in S and the constraint C are satisfiable. For all these possible decompositions, new nodes are created and stored. If no non-primitive operators can be expanded, then in lines 8-10 any operators that can be applied are applied to advance the

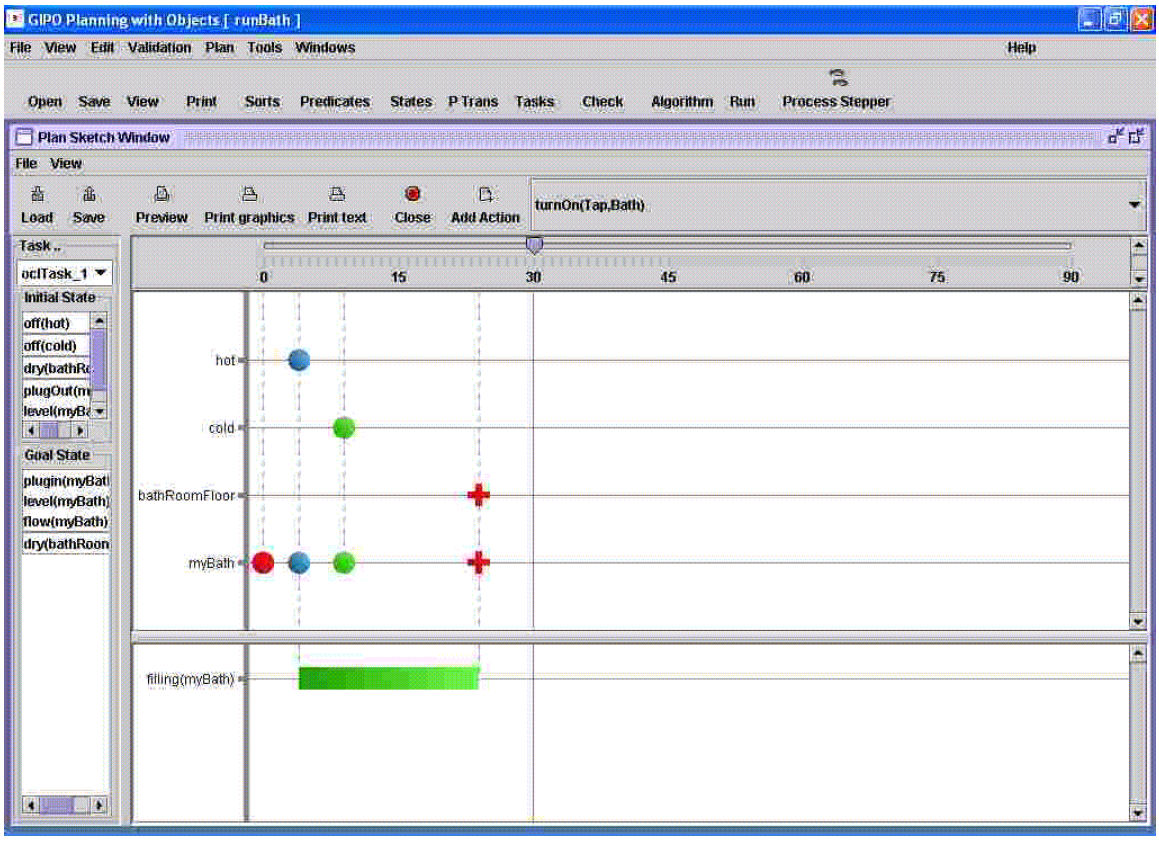

Figure 4: The Bath Filling Domain: Example of GIPO III's Plan Stepper

state. When a node cannot be expanded, it is put into a temporary store, and when all nodes are processed (or a solution is found) then the temporary store is copied back to the main store and control is given back to the main algorithm.

The procedure for simulating events and processes retracts all of the stored nodes, and with each in turn, it applies all applicable events and processes. If > 1 events occur at the same time then the order that they fire is assumed unimportant - the object states that any two firing events refer to must be independent. Similarly, if events occur during process execution, they are assumed not to interfere with the processes, that is the 'order' of simulation is not important. Finally, it is assumed that two processes that are continuously running do not directly interfere. These assumptions avoid the problem of interfering events "occurring at the same time" - which can lead to a paradox. After all the effects of applicable events and processes have been applied to a node, the time of its state is moved on an instant (signified by "Delta") and the node is stored.

The main loop progresses until expand-plan() is in a position to fully expand the abstract task into a plan containing primitive instantiated operators.

Results and Open Questions

Planning in domains with continuous processes, events and actions has long been a great challenge for AI. Real world phenomena are often outside the explicit control of plan execution and yet have to be reasoned with during plan generation. In this position paper we have outlined three tools used to formulate domain descriptions, analyse plans through simulation, and generate plans respectively, in domains with actions, events, and processes. An important aim of this work to allow the expressiveness of the *OCLplus* description to be explored and to investigate additional knowledge engineering features that may be added to the language to support the modelling and domain validation processes.

We have evaluated the tools on toy problems including an air traffic control scenario (specified in (Simpson & McCluskey 2003)), a bath domain, and an autonomous vehicle domain. The first tools (formualation and simulation) are encoded into GIPO III with *OCLplus*, and can be downloaded from http://scom.hud.ac.uk/planform/gipo. For the future we are aiming to apply the tools to a more realistic flood simulation domain. The initial work shows that (a) the GIPO tools originally designed for classical domains generalise naturally to domain descriptions in *OCLplus* (b) problems to do with the simulation are extremely complex. Depending on the choice of the time step "Delta" in both simulation and plan generation, events may fire or not fire, and processes may not run accurately or at all. The work has already raised certain fundamental questions such as: is the development of a domain-independent toolset for such

expressive domains feasible or even desirable? While continous change is expressible in domain description languages such as *OCLplus*, how can this be adequately simulated in animators and planners?

References

Bresina, J. L.; Dearden, R.; Meuleau, N.; Ramkrishnan, S.; Smith, D. E.; and Washington, R. 2002. Planning under continuous time and resource uncertainty: A challenge for ai. In *UAI*, 77–84.

Fox, M., and Long, D. 2001. PDDL2.1: An extension to PDDL for expressing temporal planning domains . In *Technical Report, Dept of Computer Science, University of Durham*.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann ISBN 1-55860-856-7.

Howey, R.; Long, D.; and Fox, M. 2004. Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In *ICTAI*, 294–301.

McCluskey, T. L., and Simpson, R. M. 2005. Towards An Algebraic Formulation of Domain Definitions using Parameterised Machines. In *Proceedings of the 24th UK Planning and Scheduling Workshop (PLANSIG-2005), City University, London*.

McCluskey, T. L.; Liu, D.; and Simpson, R. M. 2003. GIPO II: HTN Planning in a Tool-supported Knowledge Engineering Environment. In *The Thirteenth International Conference on Automated Planning and Scheduling*.

Nau, D.; Cao, Y.; Lotem, A.; and Munoz-Avila, H. 2000. SHOP and M-SHOP: Planning with Ordered Task Decomposition. Tech. Report CS TR 4157, University of Maryland, College Park, MD.

Simpson, R. M., and McCluskey, T. L. 2003. Plan Authoring with Continuous Effects. In *Proceedings of the 22nd UK Planning and Scheduling Workshop (PLANSIG-2003), Glasgow, Scotland*.

Finding Inter-Related Plans

Biplav Srivastava
IBM India Research Laboratory
IIT, New Delhi 110016, India
sbiplav@in.ibm.com

Subbarao Kambhampati*
Arizona State University
Tempe, AZ, USA 85287
rao@asu.edu

Minh Binh Do
Palo Alto Research Center
Palo Alto, CA 94304
minhdo@parc.com

Tuan A. Nguyen
University of Natural Sciences
Ho Chi Minh, Vietnam
natuan@fit.hcmuns.edu.vn

Abstract

In many planning situations, a planner is required to return a set of plans satisfying the same goals which will be used by the external systems collectively. The external systems can specify the desired inter-relationships among the returned plans (e.g., diverse plans, similar plans, non-dominated plans) and the task of the planner is to return a set of plans which will meet these requirements. As an example, in adaptive web services composition, the web service engine wants to have a set of diverse plans/ compositions such that if there is a failure while executing one composition, an alternative may be used which is less likely to be failing simultaneously. In this paper, we investigate the problem, propose functions for defining similarity among plans and propose methods to find sets of inter-related plans.

Introduction

A typical AI planner takes as input the specifications of the initial and goal states and the set of available actions, and finds a plan that will satisfy the goals by efficiently searching in the space of possible states configurations or action orderings (plans). In many planning situations, a planner is required to return not one but a set of plans satisfying the same goals which will be used by the external systems collectively. The external systems can specify the desired inter-relationship among returned plans (e.g., diverse plans, similar plans, dominated plans) and the task of the planner is to return a set of plans which meet these requirements.

As an example, in adaptive web services composition, the web service engine wants to have a set of diverse plans/ compositions such that if there is a failure while executing one composition, an alternative may be used which is less likely to be failing simultaneously. However, if a user is helping in selecting the compositions, the planner could be first asked for a set of diverse plans and when she selects one of them, the planner is next asked to find plans that are similar to the selected one. Another example is using planning for intrusion detection (Boddy *et al.* 2005), where the aim is to detect as many ways of possible intrusion as possible where an intrusion attack is represented as a plan. A third, more general example involves any complex planning situation where the user is interested in optimizing multiple and possibly conflicting objectives, to generate a set of desired plans.

Existing planners, that are designed to find single solution plans, are not well suited for this problem. Even though many of the planners are capable of outputting multiple solutions by continuing their search beyond the first solution, they cannot guarantee any desired relations between the plans that are output.

To find inter-related plans, we need to be able to (1) define distance measures between plans and (2) modify existing planners so that they can use this distance measure to generate sets of inter-related plans. Similarity and diversity are examples of such inter-relationships for plans.

There has been very little work on this problem in planning. Hebrard et al 2005 solve the problem of similar/ dissimilar solutions for CSPs. If we consider their work for planning, since a planning problem of finite length can be compiled as a CSP problem, their results are the lower bounds for finding similar or diverse plans.

Our major contributions in the paper are:

- We formalize the problem of finding diverse/ similar plans by extending previous formulations for CSPs.
- We introduce useful bases and measures for plan distance. We show that different measures can give drastically different picture about inter-plan relationships.
- We discuss some preliminary work on effective solutions to the proposed problems.

We start by formalizing the problem and then propose a series of plan similarity function. Next, we propose methods to find inter-related plan and present initial results about their effectiveness. We then explore the problem with hierarchical plans. We end with discussion on related work and provide pointers for future work.

Problem Statement

At its simplest, a planning problem PP is a 4-tuple $\langle P, I, G, A\rangle$ where P is the set of predicates, I ($\subseteq P$) is the complete description of the initial state, G ($\subseteq P$) is the partial description of the goal state, and A is the set of executable (primitive) actions. A specification of an action consists of preconditions ($A_i^{pre} \subseteq P$) and postconditions ($A_i^{post} \subseteq P$).

*Kambhampati's research is supported in part by the NSF grant IIS-0308139 and the ONR Grant N000140610058.

A plan for PP is an action sequence S_j, such that if S_j is executed in I, the resulting state of the world would contain (entail) G. It is a 3-tuple $\langle T, O, \mathcal{ST} \rangle$ where: T is the set of steps in the plan; T contains two distinguished step names t_0 and t_∞. $\mathcal{ST}$ is a symbol table, which maps step names to actions. (Note that multiple steps can be mapped to the same action.) The special step t_0 is always mapped to the dummy operator `start`, and similarly t_∞ is always mapped to `finish`. The effects of `start` and the preconditions of `finish` correspond, respectively, to the initial state and the desired goals of the planning problem. O is a partial ordering relation over T.

As an example, suppose a person in Las Vegas (LV) wants to plan a weekend and is considering to visit one or more of the Disneyland (DL) in Los Angeles, his friend in San Francisco (SF) or an event at San Jose (SJ), based on the cost of each choice and its relative utility. In Figure 1, two possible plans S_1 and S_2 are given.

We also see in the Figure that there can be different representations for a plan. For example, plans can be hierarchical consisting of non-primitive actions (tasks) which can be decomposed further into primitive/ executable actions or other non-primitive actions by one or more reductions. In the example, we had considered *Travel* actions as primitive. If we consider them as non-primitive, plans are trees of AND-OR nodes. In Figure 1, S_3^1 and S_3^2 are two possible reductions of the non-primitive action *Travel* between LV and DL. We will focus on plans with primitive actions but we note that plans in alternative representations could also be compared.

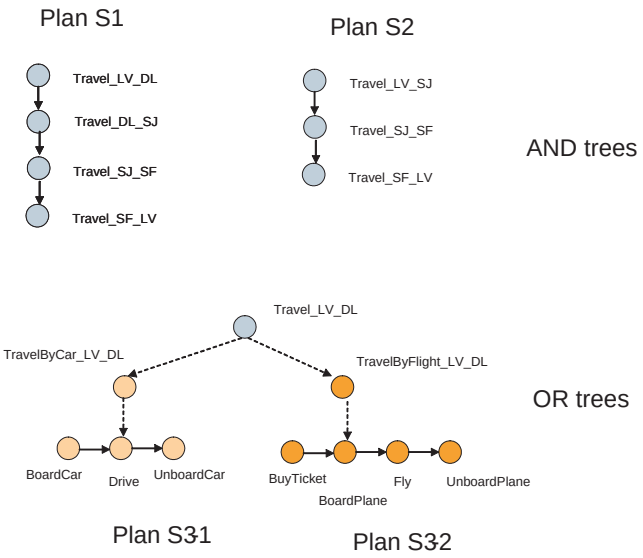

Figure 1: Examples of plans.

Let $\delta(S_i, S_j) \rightarrow [0, 1]$ denote a distance function between a pair of plans. A value of 0 represents complete similarity of plans while 1 represents complete diversity. Following the convention of (Hebrard *et al.* 2005), we define *max(*δ*,* **S***)* = $\max_{S_i,S_j \in \mathbf{S}} \delta(S_i, S_j)$ and *min(*δ*,* **S***)* = $\min_{S_i,S_j \in \mathbf{S}} \delta(S_i, S_j)$. Also, we define *max(*$\delta$*,* **S***,* S_j*)* = $\max_{S_k \in \mathbf{S}} \delta(S_j, S_k)$ and *min(*δ*,* **S***,* S_j*)* = $\min_{S_k \in \mathbf{S}} \delta(S_j, S_k)$. In Table 1, we summarize the notations that are followed and Table 2 lists the various problems to find inter-related plans. The first 3 problems, *d*DISTANT*k*SET, MAXDIVERSE*k*SET and MAX*d*DISTANTSET (and their respective *close* variants) are planning adaptations of offline CSP problems in (Hebrard *et al.* 2005) while MOSTDISTANT is adaptation of their online problem[1]. We also introduce the *nNEAR* variations of the offline problems which takes the planning problem and a reference (previous) plan as input and requires that all returned plans be close to the reference plan.

Notation	Description
PP	A planning problem
Plan(PP)	Set of all plans of PP
S_i, S_j	A plan for PP ($\subseteq$ *Plan*(PP))
S, S'	Sets of plans

Table 1: Notation used in the paper.

Problem	Description
*d*DISTANT*k*SET (resp. *d*CLOSE*k*SET) Input: PP; Output: S	Find S with $S \subseteq$ *Plan*(PP) $\mid S \mid = k$ and *min(*δ*,* **S***)* $\geq d$ (resp. *max(*δ*,* **S***)* $\leq d$)
MAXDIVERSE*k*SET (resp. MAXSIMILAR*k*SET) Input: PP; Output: S	Find S with $S \subseteq$ *Plan*(PP), $\mid S \mid = k$ and for all $S' \subseteq$ *Plan*(PP), $\mid S' \mid = k$, *min(*δ*,* **S***)* $\geq$ *min(*δ*,* S'*)* (resp. *max(*δ*,* **S***)* $\leq$ *max(*δ*,* S'*)*)
MAX*d*DISTANTSET (resp. MAX*d*CLOSESET) Input: PP; Output: S	Find S with $S \subseteq$ *Plan*(PP), *min(*δ*,* **S***)* $\geq d$ (resp. *max(*δ*,* **S***)* $\leq d$), and for all S' with *min(*δ*,* **S'***)* $\geq d$ (resp. *max(*δ*,* **S'***)* $\leq d$), $\mid S \mid \geq \mid S' \mid$
MOSTDISTANT (resp. MOSTCLOSE) Input: PP, S; Output: S_j	Find S_j with $S_j \in$ *Plan*(PP)-S, such that for S_k with $S_k \in$ *Plan*(PP)-S, *max(*δ*,* S*,* S_j*)* $\geq$ *max(*δ*,* S*,* S_k*)* (resp. *min(*δ*,* S*,* S_j*)* $\leq$ *min(*δ*,* S*,* S_k*)*)
*n*NEAR*d*DISTANT*k*SET (resp. *n*NEAR*d*CLOSE*k*SET) Input: PP, S_k; Output: S	Find S such that it is *d*DISTANT*k*SET (resp. *d*CLOSE*k*SET) and $\delta(S_i, S_k) \leq n$ for all $S_i \in S$
*n*NEARMAXDIVERSE*k*SET (resp. *n*NEARMAXSIMILAR*k*SET) Input: PP, S_k; Output: S	Find S such that it is MAXDIVERSE*k*SET (resp. MAXSIMILAR*k*SET) and $\delta(S_i, S_k) \leq n$ for all $S_i \in S$
*n*NEARMAX*d*DISTANTSET (resp. *n*NEARMAX*d*CLOSESET) Input: PP, S_k; Output: S	Find S such that it is MAX*d*DISTANTSET (resp. MAX*d*CLOSESET) and $\delta(S_i, S_k) \leq n$ for all $S_i \in S$

Table 2: Different instances of the inter-relationship aware planning problem.

Distance Measures

In this section, we motivate different bases for comparing plans, the different methods of comparing plans, and propose useful plan distance functions.

Different Bases for Plan Comparison

At the heart of the problem of finding inter-related plans is the issue of defining criteria by which two plans are compared. A plan can be characterized by:

1. Actions that are present in the plan
2. Its behavior where the behavior represents the set of states that an execution of the plan will take
3. Causal chains that support the different goals achieved by the plan. They represent a middle-ground between actions and states by encoding how actions contribute to the goal states being achieved.

[1]We have converted all the decision problems to seek their solutions.

Basis	Pros	Cons
Actions	Does not require problem information	No problem information is used
States	Not dependent on any specific plan representation	Needs an execution simulator to identify states
Causal chains	Considers causal proximity of state transitions (action) rather than positional (physical) proximity	Requires domain theory

Table 3: The pros and cons of different bases to characterize plans.

These different criteria for characterizing plans can also serve as the basis for different ways of comparing plans. Table 3 gives the pros and cons of using the different basis. We note that if actions in the plans are used as the basis for comparison, no problem or domain theory information is employed. If plan behaviors are used as the basis for comparison, the representation of the actions that bring about state transition becomes irrelevant since the actual states that an execution of the plan will take is considered. Hence, we can now compare plans of different representations, e.g., 4 plans where the first is a deterministic plan, the second is a contingent plan, the third is a hierarchical plan and the fourth is a policy encoding probabilistic behavior. If causal chains is used as the basis for comparison, the causal proximity among actions is now considered rather than just physical proximity in the plan. But it requires the domain theory to be available.

Different Ways for Computing Comparison

After a basis for plan comparison is chosen, there can be different ways of using its characterizer to derive distance functions. The analogy we use is to string comparison where inter-relationship among characters in the string are used to define distance functions. Without loss of generality, assume that we are interested in action based comparison of plans.

One way to measure distance between plans is to consider plans as sets of actions and string similarity functions that depend on characters sets. This view ignores the absolute position of an action in the plan string and cares only about the presence or absence of an action in the plan.

Set-difference based Distance Computation We can also use the set-difference measure between plans. Here, the distance between plans S_i and S_j is measured as the number of actions that occur in one plan but not the other. This measure is used in (Myers 2005; Fox *et al.* 2006).

$$\delta_1(S_i, S_j) = \frac{\mid S_i - S_j \mid + \mid S_j - S_i \mid}{\mid S_i \mid + \mid S_j \mid} \quad (1)$$

Neighbourhood-based Distance Computation We can also consider the ordering of the actions in the plan (characters in the string). Let S_i and S_j be broken into substrings **P** = $P_1, ..., P_K$ and **Q** = $Q_1, ..., Q_L$. Then neighbourhood similarity functions follow the general pattern as follows.

$$\delta_*(S_i, S_j) = \frac{1}{K} \sum_{i=1}^{K} min_{j=1}^{L} \delta^{'}(P_i, Q_j) \quad (2)$$

Name	Basis	Computation
δ_1	Actions	Set-difference
δ_2	Actions	Prefixes Neighbourhood
δ_3	States	Set-difference
δ_4	States	Prefixes Neighbourhood
δ_5	Causal Chains	Set-difference
δ_6	Causal Chains	Prefixes Neighbourhood

Table 4: A spectrum of distance functions based on different bases and way of computations.

where $\delta^{'}$ refers to some secondary similarity function. Let $\delta^{'}$ be δ_1 in the remainder. We give the distance function based on prefixes and more can be proposed based on how different substrings are created.

Prefixes-based Distance Computation **P** and **Q** contains prefixes of S_1 and S_2, respectively.

$$\delta_2(S_i, S_j) = \delta_*(S_i, S_j) \quad (3)$$

In Table 4, 6 distance functions are presented which use 3 different bases and 2 different ways of computation. More distance functions can be derived by extending any of the two dimensions.

Action	Preconditions	Effect
A1	p1	g1
A2	p2	g2
A2'	p2, g1	g2
A3	p3	g3
A3'	p3, g2	g3

Plan	Goal	Causal Chains
S6-1, S6-2	g1	I-p1-A1-g1
	g2	I-p2-A2-g2
	g3	I-p3-A3-g3
S6-3	g1	I-p1-A1-g1
	g2	I-p1-A1-g1,p2-A2'-g2
	g3	I-p1-A1-g1,p2-A2'-g2-p3,g2-A3'-g3

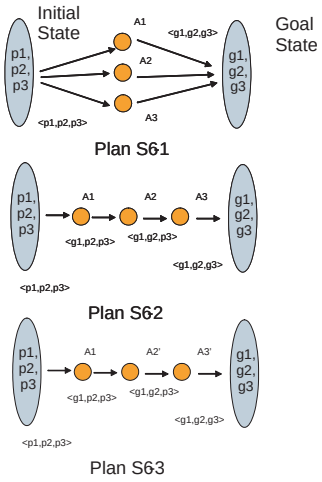

Figure 2: Three different plans for an example problem. The significant fluents of the states after every action is shown in $\prec$ ... $\succ$. The domain description and causal chains in the 3 plans are also given.

Example: Comparing Plans by Different Bases

In Figure 2, three plans are shown for a planning problem where the initial state is $\langle p_1, p_2, p_3 \rangle$ and the goal state is $\langle g_1, g_2, g_3 \rangle$. Plans S6-1 and S6-2 have the same actions but different structures. S6-1 has parallel actions while S6-2 has them in sequence. The plan S6-3 has A_1 like the other plans but all other actions are different ($A_2^{'}$ and $A_3^{'}$). However, it also achieves the same goals.

An action based plan comparison method which uses prefix-neighbourhood based distance computation would find S6-1, S6-2 and S6-3 to be all different. This is because

all the three plans have different sets of action prefixes. If instead, the action information is used with set differencing, S6-1 and S6-2 would be found identical.

A state based comparison method which uses any of the given computation choice would find S6-2 and S6-3 to be identical, and both of them to be different from S6-1. This is because the states after every transition in S6-2 and S6-3 are identical. S6-1, on the other hand, has (trivially) the same first and last states but no intermediate states.

A causal link based comparison method which uses set differencing would find S6-1 and S6-2 to be the same while S6-3 as different. The causal chains for all the goals are shown in the figure.

Solution Methods

We start by noting that one straightforward approach to generate k multiple inter-related plans is to make existing planners run further after finding the first solution. As new solutions are generated, their distance from the already selected plans can be assessed and used to decide whether they should be added to the selected set.

A more efficient alternative would be to bias the planners' search process so it progresses towards plans that are likely to meet both the solution quality and inter-releatedness constraints. There are broadly two ways of achieving this: One idea is to bias the search for the latter plans so as to satisfy the inter-relatedness constraints. In the case of heursitic search planners such as FF, this can be accomplished by modifying the heuristic to reflect the inter-relatedness constraints. For example, if we are interested in finding maximally dissimilar plans, we can penalize plans similar to the ones already selected. The technical challenge in implementing this approach would be making the heuristics sensitive to inter-relatedness constraints.

The heuristic search approach outlined above is "greedy" in the sense that the seed plans that we have already committed to could force us into a sub-optimal overall set of plans. The second idea is thus to search simultaneously for the k solutions. One avenue for doing this is to model planning as constraint satisfaction (c.f. (Do & Kambhampati 2001)), and adapt the technique proposed in (Hebrard *et al.* 2005) for simultaneously searching for k inter-related CSP solutions.

As of this writing, we have experimented with one specific implementation of each of the above ideas. We will describe the techniques implemented and provide preliminary results. We are currently in the process of completing a more careful investigation of the comparative advantages of these approaches.

Compiling Planning as CSP and Solving for Exact Diverse Plans

The GP-CSP planner(Do & Kambhampati 2001) is a Graphplan based planner that converts Graphplan's planning graph into a CSP encoding, and solves the CSP encoding using standard CSP solvers. Here, the variables correspond to the predicates that have to be achieved at a level and its possible values are the actions that can support the predicates. Constraints encode the relationship (e.g., mutual exclusions) among predicates and the relationship among the supporters of the predicates.

Similar to the way Hebrard et. al 2005 solved their *d*DISTANT*k*SET/*d*CLOSE*k*SET problem by reformulating it as a new CSP, we solve the same problem with different distance measures by making k copies of each planning encoding. Each encoding is created using GP-CSP planner and the k copies are connected to each other using global constraints. Due to the way the CSP library is used in conjunction with the planning graph structure to solve the planning encoding, there are some complications. The details of our approach are:

- As opposed to creating k(k-1)/2 special variables to represent the distances between each pair of copies, we create k(k-1)/2 global constraints connecting them. If each copy has n variable, then this constraint involves 2n variables from each of k(k-1)/2 possible pairs of k copies[2]. Each global constraint between the ith and jth copies ensures that two plans represented by the solutions of those two copies will be at least/most d diverse/similar to each other.
- Because the CSP library used in GP-CSP uses implicit constraint representation, we implement special constraint checking routine to check those k(k-1)/2 constraints. Those routines are called upon by the normal forward checking and arc-consistency checking procedure inside the default solver. In the future, we plan on implementing special consistency checking techniques to deal more efficiently with those global constraints.

Due to the special planning encoding in GP-CSP and the distance measure defined earlier in this paper, there are substantial differences between how each global constraint is satisfied between traditional CSP encoding as in (Hebrard *et al.* 2005) and our encoding. In our encoding, facts represent variables and actions represent values. A given action a can represent different values in domains of different variables. For example, if there are two variables x_1 and x_2 and their current assignments in the first copy are $\{x_1 \rightarrow v_1, x_2 \rightarrow v_2\}$ and in the second copy, they are: $\{x_1 \rightarrow v_2, x_2 \rightarrow v_1\}$, then in traditional CSP, the distance between two sets of assignments would be 2. However, the value v_1 of x_1 and v_1 of x_2 may represent the same action instance, also v_2 of x_1 and v_2 of x_2. Therefore, the distance between those two set of assignments in our planning encoding can be 2, 1 or even 0.

Thus, when each global constraint is called upon to check if the distance between two copies is within/over a predefined value d, we first have to map each set of assignments to an actual set of actions. Then, we compare the action sets (not the variable assignments) to decide if the two copies satisfy the global constraint defined by the distance measure. This process is done by mapping each variable $\rightarrow$ value into action using a call to the planning graph, which is outside but works closely with the general purpose CSP solver in GP-CSP.

[2] An alternative approach would be to create only one global constraint involving k*n variables from all k copies.

Problem	k	d	Time (in sec)	Dist.(Min, Max, Avg)
prob002-rocket-a	2	0.1	2.45	(0.154, 0.154, 0.154)
		0.2	6.72	(0.862, 0.862, 0.862)
		0.8	6.79	(0.862, 0.862, 0.862)
prob002-rocket-a	3	0.05	11.1683	(0.154, 0.862, 0.626)
		0.1	10.88	(0.154, 0.862, 0.626)
prob004-log-a	3	0.05	7.86	(0.054, 0.203, 0.151)
		0.1	24.11	(0.197, 0.698, 0.525)
		0.15	21.04	(0.197, 0.698, 0.525)
		0.2	19.50	(0.209, 0.701, 0.536)

Table 5: *Initial results of GP-CSP in* d*DISTANT*k*SET*

Table 5 presents the results of GP-CSP on some logistics problems as run on a Pentium-3 667Mhz with 256MB RAM. We see that this approach can give diverse plans effectively. The last column shows the diversity in the returned plans using δ_1 and that it is greater than minimum diversity needed for the problem (d). We also found that with higher k and d, the problems take longer to solve, as expected.

Heuristic Approach for Approximate Diverse Plans

In heuristic state space planning, a search framework like A* is used to find plans driven by heuristics that measure the progress to goals. Specifically, the cost of a search node is measured by:

$$f(s_i) = g(s_i) + w * h(s_i) \quad (4)$$

where g is the cost to achieve the current node starting from the initial search state, h is the heuristic estimate of the effort to achieve the goals and w is a weighing function. In measuring h, heuristics derived from the relaxed planning graph (RPG) have been found to be very effective(Nguyen, Kambhampati, & Nigenda 2002).

The RPG heuristic estimate can be biased towards plans that use as many of the actions already supporting other goals as possible. To do this, the planner will now take as input not just the goals to be supported, but also the set of actions already committed to in previous plans. RPG heuristic can also be biased to find plans that do not share many actions with another plan. Thus, the relaxed plan extraction process will be biased to avoid actions that are in the input plan. This approach works as long as we have similarity measures that are dependent on action presence and not on the relative position of the actions.

We implement this idea by using $h'(s_i)$ as defined below instead of $h(s_i)$.

$$h'(s_i) = h(s_i) + w_\delta * \delta_j(s_i, S_0) \quad (5)$$

Hence, we increase the heuristic values with a weighted factor accounting for the distance between the partial plan and the reference input plan. δ_j can be any distance measure including the ones defined earlier. If w_δ is positive, search nodes close to the reference plan (S_0) get priority over other nodes. If w_δ is negative, search nodes away from neighbourhood of the reference plan get priority. Given a distance function, one can start from a reference plan and control the relationship of the subsequent plan by using appropriate w_δ. The new plan can be added to the reference plans set and more plans generated appropriately related to it.

Problem	w_δ	w_k	Time (in sec)	Dist. (Min, Max, Avg)
bw-prob-4-0	-100	0	2.08	(0.334, 0.334, 0.334)
	10	100	2.04	(0.0, 0.0, 0.0)
	-10	-100	2.22	(0.156, 0.318, 0.249)
lilprob-4-0	-10	-10	1.77	(0.143, 0.334, 0.242)
	100	-100	1.68	(0.0, 0.0, 0.0)
	-100	-100	1.73	(0.0, 0.5, 0.334)

Table 6: *Initial results of Planner4J RPG for k=3*

The above approach would work regardless of whether the h was obtained from a relaxed plan of RPG (e.g., AdjSum2 heuristic) or not (e.g., max heuristic). We can also affect the relaxed plan extraction process for obtaining h. Consider the AdjSum2 heuristic where the value of a state is estimated by the length of the relaxed plan to reach the goal and an interaction factor derived from the maximum interaction among the predicates in the state.

$$h_{AdjSum2}(s_i) = len(RP(s_i)) + max_{p,q \in s_i} \Delta(p, q) \quad (6)$$

$$len(RP(s_i)) = len(RelPlan(s_i) + w_k * \delta_k(RelPlan(s_i), S_0)) \quad (7)$$

Specifically, at each step of the extraction of the relaxed plan, we prefer actions that are not present in other plans. We give a cost metric to each action in the PG and the cost is dependent on how many of the other plans these actions already support. Then we can use the costs to extract the relaxed plan.

We have implemented both the methods in the Planner4J family of Java planners and they together seem to give the best results. Table 6 presents the results on a sample of blocksworld and logistics problems as run on a Pentium-M 1.5GHz with 1.25GB RAM under Windows XP with w=5. We see in lilprob-4-0 that only w_k could not lead to diversity in the solution.

Related Work

Although the need for finding similar or different plans has been noticed in the past, there has been little concrete work on formalizing and solving the problem. Researchers including Tate (Tate, Dalton, & Levine 1998) and Myers (Myers 2005) have articulated the need for finding dissimilar plans. Myers, in particular, allows evaluating the plan similarity, but does not seem to provide a way of generating dissimilar plans efficiently. As we mentioned earlier, intrusion detection work by Boddy et. al. (Boddy *et al.* 2005) focuses on finding multiple qualitatively different plans for a problem. However, they coerce a traditional planner (Metric-FF) to generate multiple plans, and filter them out in a post-processing phase. Boddy et. al. acknowledge the need for a technique that takes inter-relatedness constraints into account during search more actively. The problem of finding similar plans has been considered in the context of replanning. A recent effort in this direction is (Fox *et al.* 2006), which shows how a local search planner called LPG can be modified to produce a plan that is similar to a reference plan.

Finally, Linden et. al. (Linden, Hanks, & Lesh 1997) motivate the need for finding related plans comprising a pareto set in the context of a travel planning scenario.

Outside of planning, we have already mentioned the connections to the work in CSP community in finding similar/dissimilar solutions. The challenges in finding interrelated plans also bears some tangential similarities to the work in information retrieval on finding similar or dissimilar documents (c.f. (Callan & Minka 2002)).

Conclusion and Future Work

In this paper, we investigated the problem of finding interrelated plans. We formalized the problem of finding diverse/similar plans by extending previous formulations for CSPs. We looked at the different bases for comparing plans, the different methods of computing comparison, and proposed useful plan distance functions. We conducted preliminary experiments with a CSP based exact approach and a heuristic based approximate approach to generate diverse plans. In future, we intend to implement more approaches and run extensive experiments.

References

Boddy, M.; Gohde, J.; Haigh, T.; and Harp, S. 2005. Course of action generation for cyber security using classical planning. In *Proc. ICAPS*. AAAI.

Callan, J., and Minka, T. 2002. Novelty and redundancy detection in adaptive filtering. In *Proc. SIGIR*. ACM Press.

Do, M. B., and Kambhampati, S. 2001. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *AI* 132(2):151–182.

Fox, M.; Gerevini, A.; Serina, I.; and Long, D. 2006. Plan stability: Replanning versus plan repair. In *Proc. ICAPS*.

Hebrard, E.; Hnich, B.; O'Sullivan, B.; and Walsh, T. 2005. Finding diverse and similar solutions in constraint programming. In *Proc. AAAI*.

Linden, G.; Hanks, S.; and Lesh, N. 1997. Interactive assessment of user preference models: The automated travel assistant. In *Proc. UM*.

Myers, K. 2005. Metatheoretic plan summarization and comparison. In *Proc. ICAPS WK. Mixed-initiative Planning and Scheduling*.

Nguyen, X.; Kambhampati, S.; and Nigenda, R. 2002. Planning graph as the basis for deriving heuristics for plan synthesis by state space and csp search. In *AI*.

Tate, A.; Dalton, J.; and Levine, J. 1998. Generation of multiple qualitatively different plan options. In *Proc. AIPS-98, Pittsburgh*. AIAI.

Planning process from a user perspective *

Oscar J. Garcia Perez, Francisco C. Palao Reines, Juan Fernandez Olivares, Luis Castillo Vidal, Tomas Garzon Hervas
oscar,palao,faro,l.castillo,tommusc @decsai.ugr.es
Dpt. Computer Science and Artificial Intelligence, Univ. of Granada (Spain)

Planning researchers are used to think about how to solve a combinatorial problem like planning efficiently. During the history of this field, algorithms, heuristics and techniques with strong formal foundations have been developed. With this advances researchers have tried to adapt their algorithms to solve real life problems, and have realized that the planning algorithm is only a small part of a much more complex system. When a planning system is introduced in the workflow of a company or an organization, this system has to communicate and collaborate with human users. So fields related to planning like knowledge acquisition, mixed initiative, machine-user interaction, plan analysis, and plan management are emerging with strong interest.

The goal of this paper is to show the experiences acquired during the development of various planning systems, specially SIADEX (Fdez-Olivares *et al.* 2006) which is a platform to assist technical staff in the forest fire fighting in the region of Andalusia (Spain), in the aim that this could help other researchers.

What users need usually is not what the planning researchers think they will need. In this sense we could distinguish two type of users, in one hand final users that will be the persons on the console using the system, and are interested on make their every day work easier, and on the other hand managers that do not unnecessarily would use our system, but are interested in improving the workflow of his enterprise or organization.

The exposition will start showing how the users see a planning system and what are the drawbacks from their point of view. Taking as a reference the SIADEX platform the importance of knowledge acquisition, correct plan management, and user interaction will be explained. Finally some final remarks will be presented.

Advantages and disadvantages of using a planning system from a user perspective

In this section we will contrast some remarkable aspects that the users balance at the moment of accepting or rejecting a planning system and are crucial from the success of the system implantation on the workflow of a company or organization.

The advantages the users see in a planning system:

- Automation and detail: The resulting plan is obtained quickly and the "tactics" of the plan are covered in detail. Users save time in their decisions.
- Correctness: The obtained plan is "correct" taking into account the knowledge encoded in the domain.
- Optimization: The plans could be optimized in time, resources, efficiency, security etc. At last this translates into a reduction of costs for the company or organization.
- Ubiquity: The plan could be accessed from any place and from any person who had permissions.
- Instant update: A change in the plan under execution, a delay of a fail, is immediately known by all the members of the organization.
- Experience capitalization: If the plans are stored correctly, the database of past plans could be analysed in order to learn from the experience or improve the workflow.
- Security: The monitoring process assures that all tasks are carried out and in any case stores the failures.

Disadvantages:

- Understanding: Experts users have difficulty in understanding synthesised plans of hundred of actions.
- Knowledge extraction: Requires willpower, dedication and time from the experts. Time and knowledge for the experts is very valuable and they are not always in good mood to share it.
- Changes in the workflow: Final users are reluctant to change their way of working. Introduction of an alien element in the workflow always requires a learning effort from the users and changes in their working routines. From the users perspective this is an inconvenient.
- Flexibility of the planner. Changes in the environment must be reflected in the plan under execution. The planner has to allow modifications from the expert user. Automated planners do not always take this into account.

*This work has been partially financed with a research contract with the Andalusian Regional Ministry of Environment (Spain)

In next sections some techniques used in the SIADEX platform in the aim to reduce the weight of these disadvantages will be presented. Also the areas in where special research effort is necessarily will be remarked.

Plan understanding

Automatically created plans are hard to understand and analyse. We need techniques to extract valuable information from the plan, specially related with the acquisition of the global structure or "strategy" underlying the plan. Tools for querying and summarising the plan are necessary. Some steps have been done (Myers 2006) but there is still an open field of research in this area. In SIADEX the problem is being solved in two ways. Firstly the way the users view the plan is very important. Different users are interested in different aspects of the plan, in our forest fire fighting domain for example "operations chief" is interested in the global strategy while "logistic chief" is more interested in shifts of the brigades and supplies. Different views of the same plan, or parts of the same plan, are presented to different human experts. Note that this views depend strongly on the domain, and the users. Secondly a generic language to query the plans is being defined under this project. Probably users will not use this language directly but could improve the process of making more advanced visualization interfaces that could show interesting statistic and strategical information. Remark that in SIADEX we have developed a translator from the XML representation of the plan to Microsoft Project XML format. This possibility to analyse, query and filter the plan in Ms Project is showing good results.

Knowledge extraction

The acquisition of information from the experts is crucial for the success of a planning system. From our point of view this is the most difficult and critical part in the development of a planning centered system, mainly because the lack of research in this area. There are general methodologies that help in the process of extracting the information, but none specific for planning. Also there is not a unique standard way to model and store planning or workflow related information. Finally there are not automatic tools that could extract the information from the users without the intervention of a knowledge engineer. Without these tools and methodologies three main problems could be identified. Firstly, the costs of extracting the knowledge and encoding the domain are high and must be done by a planning expert. Secondly, changes in the workflow of the organization imply an adaptation that only could be afforded by a knowledge engineer. And thirdly because the complexity of the models, the validation of the plans is difficult because can not be done directly by the domain experts. This problem is also related with the correct way to show the information. In SIADEX we have used UML, class, flow and process diagrams to represent workflows and the relations between the different actors. Part of the model is stored in our ontology server BACAREX but operators have been encoded in an PDDL extension.

Changes in the workflow

SIADEX is a platform of independent web services communicated through HTTP. The services could be connected or disconnected when necessary. This modular design has the advantage that different services could be inserted one by one in the workflow of the organization. In this the adaptation is easier, going from a semi-manual way of work to a fully automated system.

Flexibility of the planner

Balance between a totally user guided plan generation and a totally automated planning process must be carefully measured. In forest fire fighting domain, experts do not have time to sit down in front of the computer. A waterfall model where different goal specification, plan generation, plan reject/accept has been adopted. There are different subsets of the plan that could be planned separately, every subset depends on the decisions taken during the previous subset of the plan. The planner and the domain are carefully designed to start from a current state and a set of decided actions and generate the remaining plan. The same model could be used in plan repairing. Stop the current plan under execution, modify the state, and generate the remaining plan. As we use an HTN planner we used the expansion tree as an heuristic to guide the regeneration process searching for a new repaired plan similar to the older one in order to not to mislead the final user. There are a lack of descriptions of repairing algorithms in the literature. Researchers tend to concentrate in the planning part forgetting the repairing when in changing environments you could not make a fresh new plan, and this is probably more important.

Conclusions

All the disadvantages we have shown lead to projects that are not able to cope with this contingencies and finally are not implanted in real workflows, or in any case, that are expensive, and could only be affordable by medium/big companies or organizations. Economic costs come mainly because the lack of methodologies and tools that assist the knowledge engineer in the domain modeling. Also come from the necessity to develop and adapt customized tools for the analysis and management of the obtained plans, and even for the development of an specific planner designed to solve certain types of problems. Fortunately the interest of researchers in fields like knowledge acquisition, domain modeling or mixed initiative is growing and important advances in this areas are expected in next years.

References

Fdez-Olivares, J.; Castillo, L.; Garcia-Perez, O.; and Palao-Reines, F. 2006. Bringing users and planning technology together. experiences in siadex. In *Proceedings of the Int. Conf. on Automated Planning and Scheduling, June, 2006 AAAI press. (ICAPS'06)* . AAAI Press.

Myers, K. 2006. Metatheoretic plan summarization and comparison. In *In Proceedings of the 16th Int. Conf. on Automated Planning and Scheduling (ICAPS-06)*. AAAI Press.

Statement of Research Interest

Blazej Bulka
Department of Computer Science
University of Maryland, Baltimore County
bulka1@umbc.edu

Introduction

My current research interests can be divided into three major areas in automated planning. The first one relates to the idea of analyzing the search space by the search algorithm (planner) so that important points of the space can be identified. Another focuses on choosing appropriate representation of knowledge for the search. The third one attempts to combine different search methods or heuristics depending on local features of the search space and gained experience.

Analysis of the search space

Given a domain description, it is reasonable for the planner to analyze the domain in order to find useful hints about the structure of the search space. Such analysis can be either performed in a preliminary way or by gathering experience from multiple planning attempts. (The latter method may be used to detect properties of the search space that are very difficult to prove, or properties that are often but not always true.) Hints about the structure of the search space resulting from these two methods could be treated as a heuristic that allows a meta-planner to select an appropriate search method, or to change to a different representation (conscious shaping of the search space by the planner).

The learning approach seems to be appealing, especially given that previous research has mostly focused on analyzing the search space to identify properties that always hold. The requirement of the existence of such properties in every plan may be too strong, though. There may still be regularities in the search space, such as hub nodes, that are *usually* included in the plan. The belief that many domains (including benchmark domains for planning) have regularities is based on the successful use of heuristics to guide planning (Haslum & Geffner 2000). Additionally, Hoffmann (2003) analyzed heuristics that ignore delete lists of operations, and described phenomena that occur in the local search topology. I believe that a heuristic planner can develop better heuristics using the information mentioned earlier. For example, if a planner determines that plans in the domain often include a specific subgoal, it may be wise to start using a heuristic that adds this subgoal to the goal list right at the beginning of the planning.

Knowledge representation and shaping the search space

One way to change the shape of the search space is to change the representation of facts. My idea is partially inspired by work by Haslum and Jonsson (2000), who focused on the idea of removing redundant operators *given an initial state*. I believe that planning could be improved by removing particular effects of the operators, while still preserving correctness comparing to the plan with non-modified operators (*redundant effects*). It may also be possible to remove operators that are rarely used in plans (as learned by the planner). As a result, removing some operators would allow easier preprocessing of the domain regardless of the initial state (e.g., if it would reduce the class of a search space to a simpler one) at the expense of producing less optimal plans in terms of the plan's length.

Another way to shape the search space is to search at an appropriate abstraction level. There exist techniques that explicitly take the abstraction level into account during the search. The most notable example of such an approach is HTN planning using a predefined abstraction structure. There is also work that mentions learning of an appropriate hierarchical representation (Knoblock 1994).

Choosing an appropriate search method

Given information about the search space, either from analysis of the space or from learning during the previous searches in the space, a planner can make an informed decision to select an appropriate search method. This issue has already been partially analyzed by Veloso and Stone (1995), who present a planner that alternates between forward-chaining and backward-chaining. A similar idea could be to combine HTN planning (a top-down approach) with a more reactive approach that finds regularities in the low-level data, and decides to merge them into a higher-level structure (bottom-up approach). Deciding which of the two approaches to develop may be a challenge because none of them is perfect by itself. On the other hand, it would be beneficial to use a hybrid approach that would learn which of the search methods are most promising in the current state.

Status of the work

This paper presents research interests and thoughts arising from preliminary work done under supervision of my advisor, Prof. Marie desJardins. Currently, the ideas are far from being complete, but I plan to have the ideas further extended so that they can be used in a Ph.D. thesis proposal (planned in June 2006).

References

Haslum, P., and Geffner, H. 2000. Admissible Heuristics for Optimal Planning. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS-2000)*, 140–149.

Haslum, P., and Jonsson, P. 2000. Planning with Reduced Operator Sets. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS-2000)*, 150–158.

Hoffmann, J. 2003. Where Ignoring Delete Lists Works: Local Search Topology in Planning Benchmarks. Technical Report 185, Institute for Computer Science, Albert-Ludwigs-University, Freiburg, Germany.

Knoblock, C. A. 1994. Automatically Generating Abstractions for Planning. *Artificial Intelligence* 68(2):243–302.

Veloso, M., and Stone, P. 1995. FLECS: Planning with a Flexible Commitment Strategy. *Journal of Artificial Intelligence Research (JAIR)* 3:25–52.