An Anytime Look at Task Planning

Philippe Morignot and François Charpillet
CRIN-CNRS / INRIA-Lorraine
Bâtiment LORIA, B.P. 239
54500 Vandoeuvre-les-Nancy, France
Phone: +33 83592084 Fax: +33 83413079
{morignot, charp}@loria.fr

Category: Full paper.
Word Count: 3234, excluding references.
Keywords: Intelligent Agency, Resource-Bounded Reasoning, Anytime Algorithms, Classical Planning.

Abstract

In this paper, we present an analysis aiming at making task planning interruptible. A task planner represents a testbed for implementing interruption strategies. A control search in the space of partial has the property of representing one partial plan at a time only while being complete when not interrupted. A definition of the quality of a partial plan is proposed, in addition to the statistical notion of quality as probablity of success before a deadline. Experiments and a client application are presented.

1 Introduction

In this paper, we consider the problem of interrupting a task planner and still having a useful partial plan as output. This problem happens in every domain where real time does count. Consider for example an agent facing unpredictible events from a hostile environment, while still attempting at carrying out its mission. Such an agent might have to stop its reasoning to

consider a current unusual situation — and still use the partially generated plan for its reaction.

That problem is difficult because plan generation is an NP-complete problem [Chapman, 1987]. Thus the worst case might lead a planner to perform a search in the space of partial plans, without any guarantee about the usefulness of the partial plans before completion of the search. Actually, this problem is even worsen by the fact that finding wheather a prerequisite (i.e., a goal) inside a partial plan holds is an NP-complete problem in the general case where actions are expressed in first order predicate logic [Pednault, 1986]; thus most implementations use a subset of first order predicate logic, so as to make this algorithm polynomial [Penberthy and Weld, 1992] [Chapman, 1987].

Many approaches have been proposed by A.I. researchers to tackle with this problem. First, the engineering approach proposes to carefully design the planner and the domain description so that *usual* plans are found within a typical deadline given by the environment. A more theoretical view proposes to integrate perception of the real world into the planning process *Denise Draper. Another approach proposes to study the interruptibility of algorithms, the trade-off being a loss of correctness (anytime algorithms [Dean and Boddy, 1988] [Boddy and Dean, 1989] [Zilberstein, 1996]).

This approach has been successfully applied to path planning problems [Dean and Boddy, 1988] [Zilberstein, 1996]; this does not solve the interruption problem, though. We propose to evaluate this approach in the context of task planning problems. One potential research direction involves the notion of *abstraction*, i.e., splitting an action into several levels and solving the most abstract levels first independently of the concerned action. These abstraction levels can be generated automatically [Knoblock, 1991] or manually [Washington and Hayes-Roth, 1995] and have been demonstrated to save an exponential amount of time on some domains. It is related to anytime properties in that a planner that plans by decreasing levels of abstraction might be able to return an abstract plan when interrupted. But other studies have shown that particular choice of variables can lead such a abstraction-based search to performances that are worse than with no abstraction at all [Smith and Peot, 1992]. A more systematic study is needed, this is what we propose to undertake.

In this paper, we propose several mechanisms to give the anytime property to a task planner. For this, the next section presents the planning

algorithm. Then, a definition of a plan quality is proposed to make this algorithm anytime. First empirical results are presented and a client application is presented.

2 Context

2.1 Task Planning

Generating action plans is carried out by a general-purpose non-linear planner handling partially instantiated variables. Briefly (see [Morignot, 1991] for further details), a plan is a triple (P, U, O) where P is a set of actions, U is a unification base among variables of actions, O is a temporal ordering among actions. An action is expressed as *prerequisites* (literals that hold before the action starts), *effects* (literals the truth value of which directly change after the action is performed), *preservation* conditions (literals that hold during the action execution, e.g., (non-)unification constraints). Added and retracted effects are represented uniformly by adding a sign to literals: positive for additions, negative for retractions.

Chapman's Modal Truth Criterion [Chapman, 1987] is used for finding the prerequisites that are not satisfied (deductive use of MTC) and, given one unsatisfied prerequisite, for finding which plan modification(s) to perform to make it true (inductive use of MTC). The advantages of this algorithm are (1) the simplicity of its implementation, (2) its small complexity (i.e., polynomial: $O(n^3)$ with n the number of literals in the observed plan). But weaknesses (if not bugs) have been found since then and other algorithms might be applicable for the same purposes, e.g., a reduced version of ADL [Pednault, 1986] such as the one implemented in UCPOP [Penberthy and Weld, 1992].

The planner is extended regarding locations, arithmetic equations and variables' domain without change of the action representation. See [Morignot, 1994] for further details.

2.2 Search Algorithm

The search algorithm of the planner is the algorithm that uses area-dependent information of previous section to find a goal state given an initial state (see

below for the definition of states, etc). For planning, these aread-dependent information can be summarized by the two following functions:

- `mtc(plan)` returns in polynomial time a set of unsatisfied prerequisites, according to the MTC algorithm;
- `modifications(prerequisite)` returns in polynomial time a set of descriptions of plan modifications that will make this `prerequisite` true, according to the MTC algorith;

Many candidates are possible for such search algorithms: A^* [Nilsson, 1971], RTA^* or more aggressive ones (e.g., beam search (incomplete if the number of new states is larger than the beam's width)).

A search algorithm usually (see [Nilsson, 1971] among others) manipulates notions such as:

- a `state`, which is a partial plan in this case. The initial state begging the planning problem, expressed as virtual initial action with no prerequisite and a virtual final action with no effects.
- a `successor(state)` function, which involves: (1) running `mtc(state)`, (2) choosing one unsatisfied `prerequisite`, (3) running `modifications(prerequisite)`, (4) for every obtained plan modification, applying it to `state` to get a new state, (5) put these partial plans into a list and return it.
- a `goalP(state)` function, which is succeeds when `mtc(state)` returns nil.
- an `equal-stateP(state1, state2)` function, which succeeds iff the two states given as arguments are equal. Since a plan is a triple (P, U, O), the two plans are equal iff (1) they contain the same actions, (2) the variables have similar unification constraints, (3) the actions are ordered with similar constraints.
- a `combination(new-states, old-open-states)` function that sorts the newly generated states and the previous open ones (minus the state that generated the new states) in a unique list — let us assume that the state chosen for next expansion (i.e., the best one) is the first one.

Note that the property of *systematicity* can be added to these algorithms — systematicity is the property of a search algorithm not to visit the same plan twice [McAllester and Rosenblitt, 1991]. This can be operationalized this way: suppose that an MTC algorithm applied to some plan finds that plan modifications $(m_i)_i$ are possible; suppose that the search algorithm chooses to perform plan modification m_j first; the search algorithm is systematic iff it will perform $\neg m_j$ before performing each $m_{k \neq j}$ when exploring the remaining branches $k \neq j$.

For use in the planning area, it is desirable for the search algorithm to meet the following requirements:

1. *the successors of a state are implicit*: only the transitions to the next state (i.e., plan modifications) are explicitly generated by the planner; only the current transition leads to an explicit new state;

2. *building a partial plan, although having a polynomial complexity, is still a computationally expensive mechanism*; therefore manipulating one plan only, and incrementally modifying it, seems desirable.

The search algorithm used in the planner is a best-first search — thus, the planner is complete, in addition to being correct (since only plan modifications of `mtc(plan)` are performed).

This search uses a particular encoding of the plans that are different from the current (best) one: these plans are represented by the sequence of plan modifications that lead to them from the initial planning problem — the *meta-plan*. As a result, when the search algorithm fails to improve the current plan (i.e., it gives a better rate to these other plans), it only keeps the meta-plan of the current plan and rebuilds the desired new plan from its meta-plan — either from the description of the initial plan, or from a more advanced plan, if this can be detected from the meta-plan of the new plan and from the one of the previous plan. As a consequence, the overhead related to the application of this search algorithm to the planning area is limited.

In general, such a meta-oriented best-first search is appropriate when the heuristic evaluation function is well-informed and when the *penetrance* factor is close to one (the penetrance factor is the number of states leading from the initial state to a solution, divided by the total number of states scanned

by the search algorithm [Nilsson, 1971]). In addition, the next section favors such a view towards search for the planning area.

3 Mechanism

The notion of *anytime* algorithm relaxes the discrete notion of correctness of an algorithm's output: namely, correctness is replaced by the continuous notion of quality, given that a correct output has a quality of exactly one ($q \in [0,1]$). More precisely, an algorithm is *anytime* iff the quality of its ouput increases with the alloted time [Dean and Boddy, 1988]. As a consequence, an anytime algorithm can be interrupted and will still return a result — but probably with a lower quality than if the algorithm is given more time.

We first define the quality of a partial plan and then define a statistical quality.

3.1 Definition in teh case of a Partial plan

Since for overhead purposes, a unique partial plan is carried through the search process, the anytime property of the global algorithm relies on the definition of the quality of a partial plan. Let p be a partial plan, $mtc(p)$ the set of unsatisfied prerequisites of p (according to MTC), $actions(p)$ the set of actions in p, and $modifications(p)$ the set of plan modifications improving p (according to MTC). The quality q of a partial plan p is defined by

$$q(p) = f(\#mtc(p), \#actions(p), \#modifications(p)) \qquad (1)$$

The following properties must hold for function f in equation 1:

1. f increases when the number of unsatisfied prerequisites of p ($mtc(p)$) increases;
2. f decreases when the number of actions of p ($actions(p)$) increases;
3. f decreases when the number of plan modifications of p ($modifications(p)$) increases.

A solution plan contains no unsatisfied prerequisites, whereas the initial planning problem does — thus the first property of f.

A planner can usually find several solution plans of increasing length (e.g., uselessly moving blocks back and forth in the blocks world). A smaller solution is usually considered as "better" than a longer one — thus the second property of f.

More important, the third point is based on the following theorem:

Theorem 1 *For a particular prerequisite of an action, the number of times the MTC needs to be called depends on the number of actions in the future (solution) plan only.*

Proof. (See [Morignot, 1994].) Briefly, the MTC proposes the following plan modifications: addition of a unification constraint, addition of non-unification constraint (e.g., separation [Chapman, 1987]), addition of a time ordering constraint among actions (e.g., promotion, demotion [Chapman, 1987]), addition of an action to the plan. But a solution plan is a (set of) fully instantiated and totally ordered plan(s). This means that if a prerequisite holds for all variable's instantiations and for all total orders, it holds *a fortiori* when the first three plan modifications above are applied. Therefore the only plan modification that can change the truth value of a prerequisite of a solution-plan is action addition. □

Theorem 1 mainly suggests that the MTC should be regressed on the action templates themselves (i.e., action descriptions that are copied and instantiated to generate actions actually used in a plan) in order to predict some changes made by adding actions to a plan [Knoblock, 1991]. This theorem also suggests that no new unsatisfied prerequisite can result from a plan modification different from action addition: only existing unsatisfied need being checked which in itself leads to substantial performance improvements [Morignot, 1994].

A consequence of this is that between two action additions, only unification and ordering constraints can be added to the partial plan. In other words between two action additions, the plan is modified by adding constraints to it, i.e., reducing the number of totally-ordered fully-instantiated plans that it represents; since no new unsatisfied prerequisite can be created by this process, no new plan modification cab be created either. Therefore the number of plan modifications **must** decrease, which is what is stated in the third point above associated with equation 1.

This point has been observed experimentally in [Morignot, 1994].

Two points worth being made here: First, this can be considered as introducing constraint programming in the planning area, perturbations appearing when actions are added — coupling a constraint manipulation program with a planner during these intervals of the search. Second, the number of plan modifications is **not** the branching factor of the search algorithm: the search algorithm can contain other partial plans (even implicitly through their meta-plans, see previous section) which have their own plan modifications which is counted in the branching factor of the search.

Execution What should the execution module do with a partial plan which is not a solution? If actions of the plan are executable from the initial (virtual) action, directly executing them (before the plan is correct) might be tempting. This is true under the assumption that no future development of the partial plan will add an action between the virtual initial action and the candidate action.

We will only note here that such assumptions are very commonly made while incrementally improving a partial plan. Consider for example the case where a variable of a literal of a prerequisite is instantiated to a constant present in the initial situation — namely `?x` of the prerequisite `(block ?X)` is instantiated to `A` because of the effect `(block A)` in the initial virtual action. Nothing guarantees that no action will be added before this prerequisite that will have `(block D)` as an effect. Namely, the problem of execution of a partial plan can be instantiated to the problem of *creation of constants* during the search process —- assumptions that are usually made [Morignot, 1994].

3.2 Statistical Definition

Given a deadline, the statistical quality of a planning algorithm is defined as the probability of finding a successful plan within this deadline. The random aspect of this definition comes from (1) the variability in the input, namely of the problems that can be generated inside a particular domain, and (2) the random component of the search algorith — step 2 of the `successor(state)` function of the search algorithm can generate plan modification that will be equally top rated by the evaluation function, thus leaving space for a random component. Performance profiles can then be built, following the methodology proposed in [Zilberstein, 1996].

4 Analysis and Experiments

4.1 Domain

This planner has been implemented [Morignot, 1994] to control a Nomadics 200 mobile robot [Hayes-Roth *et al.*, 1995]. This robot is capable of sensing its environment through sonars, infra-red, bumpers and a laser/CCD pair. It can act on its environment by rotating its wheels, changing their direction, rotating a turret that supports a stereo-camero for image analysis, using a voice synthetize, vertically move an arm and open/close the arm's clamp.

From the planner's perspective, sensing and acting are both represented with action descriptions. In particular, sensing might require whole actions in itself, e.g., turn the sonar on, get some data for one second and try to recognize a corridor given the actual speed of the robot.

More specifically, the agent must perform tasks requiring motion, communication and object carrying. In first scenarios, the agent assists in office work for tasks such as delivering documents, putting cans in trashcans, warning people.

4.2 Early Experiments

First experiments intend to demonstrate the feasability of the approach. The planner is initially capable of finding plans in toy domains such as the blocks' world, augmented blocks' world, or the domains found in UCPOP's distribution source files [Penberthy and Weld, 1992].

4.3 Planned Experiments

If the planner is able to find partial plans in toy domains, we envision to use it in real domains, e.g., embedding the planner in an autonomous agent to control it. The main characteristics of these domains is that (1) the action description formalism must be expressive enough to represent actions actually performed by the agent; (2) unexpected events might interrupt the agent's reasoning, e.g., planning, and force it to use a partial plan for immediate action.

For this purpose, the agent's architecture of [Hayes-Roth *et al.*, 1995] is used to structure the software that compose our agent — that agent actu-

ally controls a mobile robot. From a software engineering perspective, such rewriting brings brighter clarity to the source-code. From a functional perspective, the symbolic (cognitive [Hayes-Roth *et al.*, 1995]) level manipulates plans that typically include less than five actions and thus that are generated in less than one second in real time. For comparison, a typical action of the robot lasts several seconds, given the maximal translational speed of the robot. We envision the interruptibility of task planning as a mean to use this agent (and the embedded planner) in more complex domains where (1) duration of execution of real actions becomes close to, or less than, the current computation time required for planning[1]; (2) the complexity of domains worsens the computation time alloted to planning.

5 Conclusion

In this paper, we have presented an analysis aiming at making task planning interruptible. A task planner represents a testbed for implementing interruption strategies. A control search in the space of partial plans has been adapted to manipulate one partial plan at a time only, while still being complete when not interrupted. A quality of a partial plan is proposed, based on observations of the behavior of the planner from a software engineering perspective — in addition to the probablitic notion of quality. Early experiments and a client application seem encouraging.

Open issues are obvious and numerous: making more precise the executability of the first action of the (partial) plan exhibited by the algorithm; exhibiting a function f that experimentally fits the monotonic increase requirement of the quality of partial plan over the search.

Ongoing research involves testing the quality and planner on a wide variety of toy domains and integrating it in a real agent running in a real domain.

[1] A closer look at [Hayes-Roth *et al.*, 1995] suggest that this could be bypassed by defining more abstract behaviors at the physical level. The question then is to which extent knowledge can be put into the physical level vs. the cognitive one.

Acknowledgements

The research described in this paper is supported by Esprit III Contract ***. Thanks to the members of the RFIA Group for fruitful discussions.

References

[Boddy and Dean, 1989] Mark Boddy and Thomas L. Dean. Solving Time-Dependent Planning Problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 979–984, Detroit, MI, 1989.

[Chapman, 1987] David Chapman. Planning for Conjunctive Goals. *Artificial Intelligence*, 32:333–377, 1987.

[Dean and Boddy, 1988] Thomas L. Dean and Mark Boddy. An Analysis of Time-Dependent Planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 979–984, Detroit, MI, 1988.

[Hayes-Roth *et al.*, 1995] Barbara Hayes-Roth, Karl Pfleger, Philippe Lalanda, Philippe Morignot, and Marko Balabanovic. A Domain-Specific Software Architecture for Adaptive Intelligent Agents. *IEEE Transactions on Software Engineering*, 21(4):288–301, April 1995. Also KSL, Tech. Rep. KSL-94-11, Stanford University, September 1994.

[Knoblock, 1991] Craig A. Knoblock. Search Reduction in Hierarchical Problem Solving. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, Anaheim, CA, 1991.

[McAllester and Rosenblitt, 1991] David McAllester and David Rosenblitt. Systematic Nonlinear Planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 634–639, Anaheim, CA, 1991.

[Morignot, 1991] Philippe Morignot. *Critères de Vérité en Planification.* PhD thesis, C.S. Dept., ENST, Paris, May 1991. In French.

[Morignot, 1994] Philippe Morignot. Embedded Planning. In *Proceedings of the Second International Conference on A.I. Planning Systems*, pages 128–133, Chicago, IL, 1994.

[Nilsson, 1971] Nils J. Nilsson. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill, New York, NJ, 1971.

[Pednault, 1986] Edwin Pednault. *The Synthesis of Plans*. PhD thesis, E.E. Dept., Stanford, Palo Alto, California, 1986.

[Penberthy and Weld, 1992] Scott J. Penberthy and Dan S. Weld. UCPOP: UCPOP: A Sound, Complete, Partial-Order Planner for ADL. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning*, Cambridge, MA, 1992.

[Smith and Peot, 1992] David Smith and Mark Peot. A Critical Look at Knoblock's Abstraction. In *Proceedings of the First International Conference on A.I. Planning Systems*, College Park, MD, 1992.

[Washington and Hayes-Roth, 1995] Richard Washington and Barbara Hayes-Roth. Incremental abstraction planning for limited-time situations. In *Proceedings of the Third European Workshop on Planning Systems*, Assisi, Italy, 1995.

[Zilberstein, 1996] Shlomo Zilberstein. Optimal Composition of real-Time Systems. *Artificial Intelligence*, 1996. To Appear.