Assumption-based Planning

Damien Pellier, Humbert Fiorino
Laboratoire Leibniz (CNRS - INPG - IMAG), Equipe MAGMA
46, avenue Félix Viallet, F-38031, Grenoble
Tel: +33 4 76 57 47 76 – Fax : +33 4 76 57 50 81
Email: {Damien.Pellier, Humbert.Fiorino}@imag.fr

This paper tackles the problem of devising an intelligent agent able to find plans under partial knowledge and/or to produce plans that partially contradict its knowledge. In other words, in order to reach a goal, such an agent is able to provide a plan *which could be executed if certain conditions were met*. Unlike "classical" planners, the planning process does not fail if some conditions are not asserted in the knowledge base, but rather proposes an assumption-based plan or *conjecture*. Obviously, this conjecture must be *reasonable* : the goal cannot be considered "achieved" and the assumptions must be as very few as possible because they become new goals for the other agents. For instance, suppose that a door is locked : if the agent seeks to get in the room behind the door and the key is not in the lock, the planning procedure fails even though the agent is able to fulfill 100% of its objectives behind the door. Another possibility is to suppose for the moment that the key is available and then plan to open the door whereas finding the key might become a new goal to be delegated. To that end, we designed a planner that relax some restrictions regarding the applicability of planning operators.

The assumption-based planning principle lies on a domain independent planning mechanism, HTN (Hierarchical Transition Network). In HTN planner [1], the objective is not to achieve a set of goals but instead to perform some set of *tasks*. The input to the agent includes a set of operators similar to those used in classical planning [2] and also a set of *methods*, each of which is a prescription on how to decompose some tasks into some sets of subtasks. The agent proceeds by decomposing *non-primitive tasks* recursively into smaller and smaller subtasks, until *primitive tasks* are reached that can be performed directly by planning operators.

In HTN planner, methods and operators are applicable if and only if the preconditions of the operators or methods are unifiable with the agent's knowledge base. In order to elaborate conjectures (i.e, plans with assumptions), this constraint is relaxed. We consider that a method or an operator is always applicable even if all the preconditions do not hold. Therefore, the application of an operator or a method involves the computation of the lacking facts. This computation is based on the unification algorithm. That is, at least one substitution that makes the preconditions match with some agent's knowledge must be founded. A substitution θ is a finite set of the form $\theta = \{x_1 \rightarrow t_1, \ldots, x_n \rightarrow t_n\}$ where every x_i is a variable, every t_i is a term not equal to x_i, and $x_i \neq x_j$ for any $i \neq j$. Let θ be a substitution and p be the preconditions of an operator or a method. Then $\theta(p)$ is an expression obtained from p by replacing simultaneously each occurrence of the variable x_i with the term t_i. For each substitution computed and applied to the operator or method preconditions, we check if the preconditions are contained in the agent's knowledge base. If this is not the case, these preconditions are considered as *assumptions*. For example, let *move* describe an operator that allows to move a taxi t from a location x to y. To apply this operator, the taxi must have fuel and a passenger must be loaded:

move(t, x, y)
pre $\{at(t,x), hasfuel(t,q), isloaded(t), (q \geq 10)\}$
del $\{at(t,x), hasfuel(t,q)\}$
add $\{at(t,y), hasfuel(t,(q-10))\}$

Let the agent's knowledge be as follows:

$$\{at(cab38, downtown), isloaded(cab38), hasfuel(cab38, 10), (not(isloaded(cab74))), hasfuel(cab74, 5)\}$$

Two possible substitutions can be computed:

$$\theta_1 = \{t \rightarrow cab38, x \rightarrow downtown, q \rightarrow 10\}$$
$$\theta_2 = \{t \rightarrow cab74, x \rightarrow downtown, q \rightarrow 5\}$$

The substitution θ_1 applies to the preconditions of the *move* operator and do not produce assumptions. But in the other substitutions, triggering the operator *move* involves to make the following assumptions: (1) $at(cab74, downtown)$, (2) $isloaded(cab74)$ and (3) $hasfuel(cab74, q)$ with $q \geq 10$.

The assumption-based planning algorithm can make three different kinds of assumptions: (i) *Fact generation:* the substitution can generate literals that do not belong to the current knowledge base. This means that expressions absent from the current knowledge base are not considered as false but rather as unknown (assumption 1 in the example: *cab74* is supposed to be downtown even though this fact is not asserted); (ii) *fact negation:* if an atom in the substitution is the negation of a fact in the current base, then this fact is withdrawn and replaced by its negation. In that case, the agent knows that its reasoning contradicts its knowledge (e.g. it knows that *cab74* is not loaded but it acts as if it was loaded, see assumption 2), but it bets on its teammates ability to change the world consistently (i.e. load *cab74* before moving it); (iii) *Constraint violation:* as in fact negation, constraints can be violated (assumption 3 in the example).

Of course, the assumptions must be carefully generated: the conjecture must make the fewest assumptions. To that end, our algorithm is based on a reachable states space search. This states space is stored in a tree called the *conjecture tree*.

The algorithm can be split in two different steps: the conjecture tree expansion which represents the reachable states space and the conjecture extraction. The parameters of the algorithm represent an agent planning domain: S is the initial state (i.e., the agent's knowledge), O the operators set (i.e., the agent's skills) and T the tasks list to be done. The conjecture tree is initialized with the initial state S that represents the tree's root node.

The first step is the computation of the conjecture tree. This tree contains the different steps of the agent reasoning. Each node represents a state of the world that can be reached. More precisely, a node of the tree is a tuple $N = (S, T, w)$, where S (a set of ground atoms) is a state, T (a list of task atoms) is the task list remaining to be performed at this reasoning step and w is the node valuation. This valuation is the number of assumptions done to reach this node from the root node. The edges explicitly represent the possible transitions between the different states of the world (i.e. the methods or operators heads plus the corresponding assumptions where appropriate). The conjecture expansion stops as soon as a leaf with an empty task list T is reached.

The second step is the conjecture extraction from the conjecture tree. A conjecture is represented by a branch (i.e. a path from the root node to a leaf). For each edge of the branch, if the task is primitive then the task and its assumptions are added to the conjecture, otherwise only the assumptions are added. More precisely, a conjecture is an ordered list of couples (A, h) where A is a set of atoms (the assumptions required to fulfill h) and h a head of ground operator instance.

The conjecture tree expansion is not a simple deep first exploration: the computation of the conjecture with the fewest assumptions is equivalent to a minimization problem. In order to solve this problem, the expansion algorithm is based on the "best first" principle. The nodes stored in the conjecture tree are valuated from the number of assumptions made to reach them and the node with the weakest valuation is recursively chosen at the expansion step to expand the conjecture tree until a leaf is found.

Soundness and completeness: The algorithm tries to decompose the initial goal in an ordered list of primitive tasks. As in HTN planner such as SHOP [1], for a finite search space, the construction of the conjecture tree is sound and complete; solutions without assumptions, if they exist, are proposed first. However, our algorithm is more "greedy" than HTN planners because, when assumptions must be done, more nodes are created.

Search limitation: The number of allowed assumptions can be bounded in order to end the search process at an arbitrary limit. When the limit is set to 0, the algorithm is equivalent to classical HTN planners. This can be used to adapt our algorithm to the system capabilities and find conjecture with more and more assumptions.

Choice of a planning system: There are many different planning systems (e.g., planning based on Binary Decision Diagrams [3], heuristic search [4], constraints satisfaction [5] and so forth). However, HTN planner is well-suited for the assumptions generation due to the substitution procedure that allows to compare the agent's knowledge with the preconditions necessary to trigger an operator or a method. We investigated the possibility to generate assumptions with planners like GRAPHPLAN [6] but this turned out to be much more difficult because of the forward-chaining process and the absence of substitution procedure.

The aim of this research is to propose a complete multi-agent planning process based on the concept of proof validity that can be considered as an exchange of proposals and counter-proposals. According to [7], a correct proof does not exist in the absolute. At any time, an experimentation or a test can refute a proof. If one single test leads to a refutation, the proof is reviewed and it is considered as a mere conjecture, which must be repaired in order to reject this refutation and consequently becomes less questionable. The new proof can be subsequently tested and refuted anew. Therefore, the proof elaboration is an iterative and non monotonous process of conjectures/refutations/repairings. In planning context, each agent can refine, refute or repair the current conjecture. In particular, refining a conjecture means substituting assumptions by subplans achieving them, and refutations are linked with mutual exclusions [6]. If the reparation of a previously refuted plan succeeds, it becomes more robust but it can still be refuted later. If the reparation of the refuted plan fails, the agents leave this part of the reasoning and explore another conjecture: "bad" conjectures are ruled out because there is no agent able to push the process further. In this case, the cost of providing another conjecture is low because the agents can rely on the conjecture tree already computed and resume their exploration. Finally, as in an argumentation with opponents and proponents, the current conjecture is considered as an acceptable solution when the proposal/counter-proposal cycle ends and all the assumptions have been removed.

REFERENCES

[1] D. Nau, T. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu, and F. Yaman, "Shop2: An HTN planning system," *Journal of Artificial Intelligence Research*, vol. 20, pp. 379–404, 2003.

[2] R. Finkes and N. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 3-4, no. 2, pp. 189–208, 1971.

[3] R. M. Jensen and M. M. Veloso, "OBDD-based universal planning for multiple synchronized agents in non-deterministic domains," in *Artificial Intelligence Planning Systems*, 2000, pp. 167–176.

[4] M. B. Do and S. Kambhampati, "Sapa: A domain-independent heuristic metric temporal planner," in *Proceedings of the European Conference on Planning*, 2001.

[5] H. Kautz and B. Selman, "Unifying SAT-based and graph-based planning," in *Workshop on Logic-Based Artificial Intelligence*, J. Minker, Ed., College Park, Maryland, USA, 1999.

[6] A. Blum and M. Furst, "Fast planning through planning graph analysis," *Artificial Intelligence*, vol. 90(1-2), pp. 279–298, 1997.

[7] I. Lakatos, *Proofs and Refutations*. Cambridge University Press, 1976.