Representation and Control in IxTeT, a Temporal Planner

Malik Ghallab Hervé Laruelle
LAAS-CNRS
7, avenue du Colonel-Roche
31077 Toulouse Cedex - France
malik@laas.fr laruelle@laas.fr

Abstract

This paper presents a temporal planner, called IxTeT. It focuses on the representation and control issues, arguing for a compromise between the expressiveness and the efficiency of the search. The representation relies on a point-based reified logic, associated to multi-valued domain attributes. Hierarchical planning operators offer an expressive description, with parallelism, durations, effects and conditions at various moments of the action. Time in the input scenario enables to take into account predicted forthcoming events and to plan in a dynamic world. A compilation procedure checks the consistency of the operators specified by the user. The control relies on the use of causal-links, A_ϵ algorithm, and an extended least-commitment strategy. It uses two important procedures, called "feasibility" and "satisfiability", dealing respectively with goal decomposition and conflict resolution:

Introduction

This paper describes a planning system, called IxTeT, which is intended to be a task-level planner for a robot. We consider a robot as a programmable machine. Its planning system relies as much as possible on specific representations and modules. But at the task-level, it is also a compromise between what is explicitly given by the programmer and what is found by the machine, at preprocessing time and on-line. This trade-off requires a balance between the expressiveness of the representation and the efficiency of the on-line domain-specific and general search algorithms used.

In IxTeT time is specifically represented and dealt with, in planning operators and in domain scenarii, from which planning starts. Time in planning operators is needed to generalize partial order to concurrent action plans, and to express actions with effects and conditions occuring at various moments of their duration. Time in domain scenarii enables to take into account predicted forthcoming events and to plan in a dynamic world. The knowledge representation in IxTeT has other interesting features: multi-valued domain attributes temporally qualified into instantaneous events and persistent assertions; an ontology, relevant for planning, which classifies attributes as *rigid* or *flexible*, *contingent* or *controllable*; a hierarchy of tasks as planning operators; constraints on domain variables handled specifically. Preprocessing techniques are extensively used in IxTeT at compile-time to check planning operators and input scenarii for consistency and to translate them into more explicit and efficient representations. Causal links organize the search in the space of partial plans which is controlled by efficient search algorithms and carefully chosen heuristics.

Some of these IxTeT features are original, others are borrowed from the most advanced contributions to planning. The state of the art is indeed fairly mature. The algorithmic bases of planning have been well clarified in [3]and [14]. Work in deductive planning has shed light on the difficulty of tackling issues such as the ramification and qualification problems, and reasoning about change with uncomplete information [2] [4]. Temporal aspects in planning have been worked on by numerous authors along different lines [17] [6] [1] [16]. But there are very few systems, like O-PLAN[5] and SIPE[19], that aim at integrating into an effective compromise powerful representation and efficient algorithms. The ambition of IxTeT is to be one of those as a temporal planner.

The Timelogic temporal planner [1] relies on the interval algebra, as does TLP [16], whereas for complexity reasons IxTeT uses the time-point and restricted interval algebra. The TMM of FORBIN[6] also uses time-points as primitives, but the approach of IxTeT for managing the network of temporal constraints is more efficient. Our representation for planning operators appears to be more effective than that of Timelogic for expressing complex tasks and for simplifying planning through programming. Another planner, TRIPTIC [10], uses temporal tasks described by intervals without domain variables. Some of the above systems propose to integrate powerful constructs, such as a truth maintenance module for handling persistence [7], that we have not retained in IxTeT because of their computational cost.

The rest of the paper describes the proposed representation (section 2), then the control implemented in

IxTeT (section 3). A complex example illustrates the performances of the system.

Representation

Formalism

Time For algorithmic complexity reasons our time-map manager relies on **time-points** as the elementary primitives [8]. We consider time as a linearly ordered discrete set of instants. Intervals and relations of the restricted interval algebra, equivalent to the time-point algebra [18] can also be represented at the user level. They are translated internally into time-point constraints (we will not develop that issue here). We can handle the usual **symbolic constraints** of the time-point algebra (i.e. before, simultaneous, after and their disjunctions), as well as **numerical constraints**. The later are expressed as pairs of real numbers $[I^-, I^+]$ corresponding to lower bounds and upper bounds on the temporal distance between two points. Disjunctions of such constraints are not allowed. IxTeT maintains the consistency of the time-map through two separate networks for the two types of constraints. Algorithms have been fine-tuned for the specific needs of a planner, where there are usually fewer numerical constraints than that symbolic constraints. The later can be maintained with inexpensive linear the average complexity procedures [8].

Domain Attributes The world is described by a set of **multi-valued domain attributes**. Each attribute is a k-ary mapping from a finite domain into a finite range; both sets are explicitly declared. Multi-valued attributes are more expressive than binary relations: domain constraints such as the non ubiquity of objects are naturally handled by mapping over **unique** values. Attributes belong to the following ontology:

- **rigid** attributes (or atemporal): whose value does not change over times, they express a structural relationship between their arguments, e.g. $Is\text{-}key\text{-}of(key) \in \{door_1, door_2, \ldots\}$.
- flexible attributes (or fluents), whose value may change; they are either
  - **controllable** attributes : their change of value can be planned for, but they may also change independently of the planning system, e.g. $Position(object) \in \{room_1, room_2, \ldots\}$, or
  - **contingent** attributes : their change is not controlled by the planner , e.g. $Sun\text{-}light(location) \in \{day, night\}$.

Expected changes in flexible attributes, contingent as well as controllable, that may affect the planned activity are assumed to be known and given as input. We restrict ourselves to flexible attributes that are homogeneous, or "liquid" in the sense of [15], i.e. whose persistent value over an interval is inherited over subintervals (upward and downward). We do not consider continuous change or processes.

Events and Assertions We rely on a reified logic formalism [15] where fluents are temporally qualified by predicates such as **Hold** and **Event**. $Hold(att(x_1, \ldots) : v, (t_1.t_2))$ asserts the persistence of the value of attribute $att(x_1, \ldots)$ to v, $\forall t : t_1 \leq t < t_2$. $Event(att(x_1, \ldots) : (v_1.v_2), t)$ states that an instantaneous change of value of $att(x_1, \ldots)$ from v_1 to v_2 took place at time t. By definition the value at time t is v_2; this avoids the problem of undefined value at the transition point. The formal definition of the semantics of *Hold* is easily given with respect to an interpretation and a meaning function. This function sets a ***unique*** value at a given time point to each possible instance of each attribute. Predicate *Event* can be defined by:

$$Event(att(x_1, \ldots) : (v_1.v_2), t) \stackrel{def}{\equiv}$$
$$\exists t_1 \exists t_2 : \ (t_1 < t < t_2) \wedge Hold(att(x_1, \ldots) : v_1, (t_1.t)) \wedge Hold(att(x_1, \ldots) : v_2, (t.t_2)) \wedge (v1 \neq v2)$$

Literals in our representation are :

- temporal propositions: Event or Hold predicates qualifying flexible attributes over domain constants and variables, through temporal constants and variables,
- value predicate for rigid attributes over domain constants and variables (noted $att(x_1, \ldots) = v$),
- unary and binary instantiation constraints over domain constants and variables, of the form $(x_i = x_j), (x_i \neq x_j), or (x_i \in$ finite subset$)$, and
- temporal constraints, symbolic (time-point and/or restricted interval algebra relations) or numerical, over temporal constants and variables.

All variables are universally quantified. Conjunctions of such literals are called **indexed time tables** (acronym for **IxTeT**). In such a table, temporal constraints are gathered into the time-map; instantiation constraints into a **variable binding network**, which also manages rigid attributes. In this later network, equality constraints are propagated by transitive closure; the other constraints are propagated by an arc-consistency procedure. A temporal or instantiation constraint is said to be **necessary** in an IxTeT if it is entailed by the explicit constraints in the table, it is a **possible** constraint if the IxTeT does not entail its opposite.

Planning Operators

A **hierarchy** of operators, called **tasks**, is defined. A task is composed of: (i) a set of required conditions described by *assertions* on controllable or contingent attributes; (ii) a set of subtasks; (iii) a set of effects, in addition to those of the subtasks, described by *events* on controllable attributes; (iv) a set of temporal and instantiation constraints, binding variables of the task. This flexible representation enables the description of concurrent actions as well as combined and/or context

dependent effects into a task. It does not however allow low recursion or other complex control structure into a task. Notice that the planner itself is not hierarchical, e.g. as ABTWEAK[12].

Tasks are *deterministic* operators, without ramification effects. We assume that all attributes which are not explicitly modified by a task or its subtask stay unchanged after the application of this operator, unless specified in the input scenario as a predicted change.

Task **LOAD-ROBOT**{
Hold(Position(?robot):?room, (?start. ?end));
Hold(Light(?room):lit, (?start. ?end));
Hold(Occup(?robot):busy, (?start. ?end);
Hold(Occup(?arm):busy, (?t1. ?t2);

Task GRASP(?obj, ?arm, ?room)(?start, ?t1);
Task UNGRASP(?obj, ?arm, ?robot)(?t2, ?end);

Event(Occup(?robot):(free. busy), ?start);
Event(Occup(?robot):(busy. free), ?end);

?t1 ≤ ?t2;
?end - ?start in [1.0,2.0];
?robot ∈ ROBOTS;
Is_arm_of(?arm): ?robot;}

Figure 1: *Description of the task LOAD-ROBOT*

A task *LOAD-ROBOT* is presented in figure 1. It is important to notice the difference between our required conditions and "preconditions" in the usual state operators. For example, $Event(Occup(?robot) : (free.busy), ?start)$ expresses an effect, the robot being busy, but it also implies the condition requiring that the robot is free before the instant "*?start*". This is only true for an event. An assertion may not need to be established by an additional event. For example, assertion $Hold(Occup(?robot) : busy, (?start.?end))$ is established by an event of the task itself, and it only ensures the persistence of the fact $Occup(?robot) : busy$ during the task interval.

Our representation improves over a temporal representation where operators have a simple duration. First, the possibilities of **establishments are enlarged**. Indeed, a task can establish an assertion, which persists only *during* the task, but not before nor after. For example, if a task T_1 requires the condition $Hold(Position(?obj) :?arm, (t.t'))$, this condition can be established by the task *LOAD-ROBOT*, through the effects defined in the subtasks *GRASP* and *UNGRASP*. Task are **breakable**, unless a contrary constraint is specified: between its subtasks the planner may insert other subtasks. For example, if the assertion $Hold(Occup(?arm) : taken, (?t1.?t2))$ is removed from the task *LOAD-ROBOT*, another task may use the arm between the two subtasks. In some cases such an insertion of other tasks may be needed to make a task feasible.

Initial Plan $\mathcal{P}_{init}$

Input data, noted $\mathcal{P}_{init}$, is a scenario which describes the initial state, expected change in the world and required goals. We assume a complete information on the value of attributes at the initial time point. Expected contingent events are inserted into $\mathcal{P}_{init}$. They must be consistent, i.e., events on the same domain attribute should be totally ordered with values compatible with this order. Variables are not allowed as arguments of contingent attributes.

Finally, the user can specify some assertions and events, temporally constrained, on controllable attributes. Some of them, which stay "pending" and need to be explained, are the goals of $\mathcal{P}_{init}$. We assume that no other change occurs on controllable attributes except those given by the user and planned for.

Consistency Criterion

A partial plan $\mathcal{P}$ is possibly inconsistent if the same attribute may take two different values at the same time. In that case it contains conflicts that need to be solved. A conflict is either:

(a) $c=\langle Hold(att(x_1,\ldots,x_k) : v_1,(t_1.t_1')),$
$Event(att(y_1,\ldots,y_k) : (v_2.v_2'),t_2)\rangle,$
such-that $\mathcal{P}$ does not contain one of the constraints
$constr_{(a)} = \{(t_2 < t_1);(t_2 = t_1\ and\ v_1 = v_2);$
$(t_1' \leq t_2);(x_1 \neq y_1);\ldots;(x_k \neq y_k)\}.$

or

(b) $c= \langle Hold(att(x_1,\ldots,x_k) : v_1,(t_1.t_1')),$
$Hold(att(y_1,\ldots,y_k) : v_2,(t_2.t_2'))\rangle,$
such-that $\mathcal{P}$ does not contain one of the constraints
$constr_{(b)} = \{(t_2' \leq t_1);(t_1' \leq t_2);(v_1 = v_2);$
$(x_1 \neq y_1);\ldots;(x_k \neq y_k)\}.$

Each constraint of *constr*, named a **resolver**, if added to the partial plan $\mathcal{P}$ solves the conflict. A conflict is solvable if one of its resolver constraints is consistent with the current constraints of $\mathcal{P}$. Hence:

Consistency Criterion: A partial plan $\mathcal{P}$ is consistent iff:
(1) The temporal network is consistent,
(2) The variable bounding network is consistent, and
(3) Every conflict in $\mathcal{P}$ is solvable.

A temporal proposition is said **explained** if there is an event establisher and if there is no event clobberer between the establisher and the proposition. This last property is *naturally* expressed in our representation.

Explained temporal propositions:
An assertion $Hold(att(x_1,\ldots,x_k) : v,(t.t'))$ or an event $Event(att(x_1,\ldots,x_k) : (v.v'),t)$ are explained iff:
(1) there is an event establisher $Event(att(y_1,\ldots,y_k) :$

$(v'.v_{est}), t_{est})$, such-that necessarily $(v_{est} = v)$, $(t_{est} < t)$ and $(y_1 = x_1), \ldots, (y_k = x_k)$; and
(2) there exists an assertion $Hold(att(x_1, \ldots, x_k) : v, (t_{est}.t))$ between the establisher and the proposition.

If a proposition in a plan $\mathcal{P}$ checks only part (1) of the definition and the establishment constraints are only possible, this proposition is said to be *establishable*. The unexplained propositions of the current partial plan $\mathcal{P}$ constitute the subgoals of the plan to be solved. A **resolver** of an unexplained proposition is the list : $(Event(att(y_1, \ldots) : (v''.v_{est}), t_{est}),$
$(v_{est} = v), (t_{est} < t), (y_1 = x_1), \ldots, (y_k = x_k))$.

Notice that the procedure which checks the consistence of a partial plan runs in polynomial time ($O(n^3)$). However it is not complete since the variable binding network is checked by arc consistency. Indeed, a complete constraint propagation in this network could be more costly than the planning search, which will precisely reduce the complexity of this propagation by adding equality constraints (through establishments). A complete propagation is performed at the end, in order to ensure that the found plan is really a solution.

The compilation procedure

This procedure checks the consistency of the input data and makes it more explicit into efficient data structures. It transforms each task and the initial plan into an IxTeT structure. Elementary tasks, without subtasks, are compiled at first. Then tasks at the level just above are compiled, replacing subtasks with their corresponding IxTeT. This procedure is repeated until the top level of the tasks hierarchy. Only tasks at the highest level of the hierarchy and those explicitly specified by the programmer, are used by the planner. The others are considered as a programming facility.

The consistency of the temporal network and the variable binding network of each task and of the initial plan are checked. According to the consistency criterion, the compilation procedure then checks that each conflict is solvable.

Finally, the compilation procedure marks the unexplained temporal propositions of each IxTeT. They define the set of subgoals to be established, in the initial plan or in a task, if it is inserted in a partial plan.

The planner will take as input the IxTeT of the initial plan, and will add to it constraints or tasks. Each partial plan is itself expressed by an IxTeT structure.

Control

Starting from the initial plan $\mathcal{P}_{init}$, the search of a solution consists in solving incrementally the conflicts and subgoals, by adding tasks, temporal constraints or instantiation constraints. At each step of the search, the *consistency* of the partial plan is checked. If it is not consistent the planner backtracks. The search stops when a **solution plan** is reached, i.e. a plan $\mathcal{P}$ such that :
(i) all assertions and events are explained; and
(ii) no conflict remains in $\mathcal{P}$.

The search tree is controlled by a near-admissible algorithm A_ε. This algorithm provides a trade-off between the efficiency of the search and the quality of the found solution plan. **Causal-links** are used to avoid a too redundant search. The control relies on an **extended least commitment** strategy, which enables to choose conflicts or subgoals that have the most "constrained" resolvers, heuristically evaluated. Two procedures are used: **feasibility**, which computes and evaluates potential solutions for subgoal establishment in the partial plan $\mathcal{P}$; and **satisfiability** which computes and evaluates conflict resolution in $\mathcal{P}$.

The Global Search Procedure

GLOBAL_SEARCH

- SolutionPlans ← ∅, partial plan $\mathcal{P}$ ← Initial Plan;

WHILE(there is no solution plan $\mathcal{P}_s$ with f($\mathcal{P}_s$) < Threshold) and (pending nodes remain) DO

- 1. Subgoals with their resolvers ← FEASIBILITY($\mathcal{P}$).
- 2. Conflicts with their resolvers ← SATISFIABILITY($\mathcal{P}$).
- 3. **Choose** the most constrained conflict or subgoal (smallest K), with its set of resolvers $\mathcal{R}$
- 4. For each resolver r in $\mathcal{R}$,
  - evaluate f($\mathcal{P}$+r) ← g($\mathcal{P}$+r) + h($\mathcal{P}$+r).
  - Threshold ← (1 + ε)×(the minimal value f of all the pending nodes).
- 5.1 **Choose** the resolver r_0 with the minimal f in $\mathcal{R}$
- 5.2 For the other r of $\mathcal{R}$,
  - put ($\mathcal{P}$ + r) in the set of pending nodes.
- 6.1 $\mathcal{P}$ ← $\mathcal{P}$ + r_0
- 6.2 If $\mathcal{P}$ is solution then put $\mathcal{P}$ in SolutionPlans and
- 6.3 If ($\mathcal{P}$ is inconsistent) or (f($\mathcal{P}$) > Threshold), then
  - Backtrack to the pending node with the minimal f.

End WHILE.

- Return the solution in SolutionPlans with minimal f

Figure 2: *The global search algorithm*

This procedure is given in figure 2, and is defined from A_ε [9]. This algorithm performs a depth-first search as long as the current partial plan is *acceptable* (step 5); when it is not acceptable, backtracking occurs according to a best-first strategy (step 6). Acceptability requires the consistence of the partial plan and an evaluation above a threshold (defined at step 4), with

respect to the estimated best partial plan in the search space.

The steps 1 and 2 of this algorithm determine conflicts and subgoals. Each conflict is computed with its resolvers : the set $constr_{(a)}$ or $constr_{(b)}$ reduced to the possible constraints. Each constraint is qualified by a cost, proportional to the reduction of the interval $[I^-, I^+]$ for a temporal constraint, or to the reduction of the domains of variables for an instantiation constraint. Each subgoal is computed with its resolvers : the set of possible establishers of $\mathcal{P}$, and the new tasks which may establish it. Each establisher and task is qualified by the sum of : (i) the costs of its constraints, (ii) the cost of a causal-link proportional to its time span, and (iii) a fixed user-defined cost of the tasks, and of the new subgoals entailed by these tasks.

There are two non-deterministic steps in this procedure: at step 3, the choice of the next conflict or subgoal to be solved, and at step 5.1, the choice of a resolver. The former choice influences the order of conflicts and subgoals resolution in the search tree. It does not involve an expansion of the tree, and hence no backtracking. It has been shown that the efficiency of the planner depends notably of the order of subgoals [11]. Here we take into account subgoals *and* conflicts. For that, we extend the principle of least commitment, which requires that a constraint or a task is inserted only when it is necessary. When all conflicts and subgoals have several possible resolvers, we evaluate the set of resolvers for each conflict or subgoal by a function K:

$$K = \sum_{resolvers} \frac{1}{1 + cost(resolver) - cost_{min}}$$

where $cost_{min}$ is the resolver with the minimal cost. The choice of the conflict or the subgoal with the minimal K corresponds to a resolver with the higher difference of cost between it and the next bests, hence it reduces the chance of backtracking.

The estimate is $f = g + h$, where g is the cost from the root $\mathcal{P}_{init}$ to the current partial plan $\mathcal{P}$, and h is a heuristic, which estimates the cost between $\mathcal{P}$ and a solution plan. The cost function g is the sum of the cost of constraints and tasks inserted in the initial plan, as previously defined. The heuristic $h(\mathcal{P})$ is the minimal cost of the constraints and tasks to be added to $\mathcal{P}$ to solve all the conflicts and subgoals. This heuristic is a lower bound since it does not take into account the new conflicts and subgoals entailed by new tasks. Therefore, the algorithm provides an *ε-optimal solution*. However, this lower bound heuristic has a computational overhead and is not always the more efficient. Note that costs taken into account are not action execution costs, but only estimates computed by the planning process.

When we decide to establish a subgoal by an event in $\mathcal{P}$ or by a new task, the assertion $Hold(att(x_1, \ldots) : v, (t_{est}.t))$ is added at the step 6.1 as a causal-link, in order to protect the establishment of the subgoal in all the search sub-tree following this node.

The Feasibility Procedure

The establisher of a subgoal can be met by an event in the partial plan $\mathcal{P}$, or by a new task containing an event which can explain it. If a subgoal requires a new task –ie $\mathcal{P}$ contains no possible establisher–, the feasibility procedure develops a tree of subgoal decomposition, in order to estimate the tasks which may solve the subgoal. Indeed, a look-ahead is necessary for weighting carefully the choice of a new task, since it may cause new subgoals and many other conflicts.

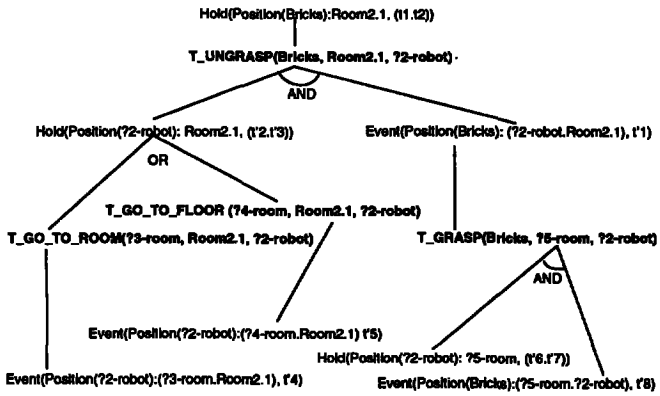

Figure 3: *Tree of Subgoal Decomposition*

The subgoal decomposition consists in expanding an AND/OR tree, where the nodes AND are tasks, and the nodes OR are subgoals. Figure 3 presents the decomposition tree of the subgoal $Hold(Position(Bricks) : Room2.1, (t1.t2))$. The decomposition begins with the search of the tasks $T_1, \ldots, T_n$ which establish the root node. Then, an establisher for each unexplained temporal propositions of each task T_i is searched for in the partial plan $\mathcal{P}$. If a proposition is establishable, it is not developed. If it is not establishable, it is developed in turn.

The expansion of the AND/OR tree, limited to a fixed depth, is controlled by an AO^* algorithm, which always develops the sub-tree with minimal cost. The cost of a task is the sum of its unexplained propositions. The cost of a subgoal is the minimal cost (as defined above) of the children tasks or of events in $\mathcal{P}$ which establish it. A task cannot be an establisher if one of its subgoal cannot be established : an infinite cost is assigned to it.

This procedure provides only an *estimate* of the cost

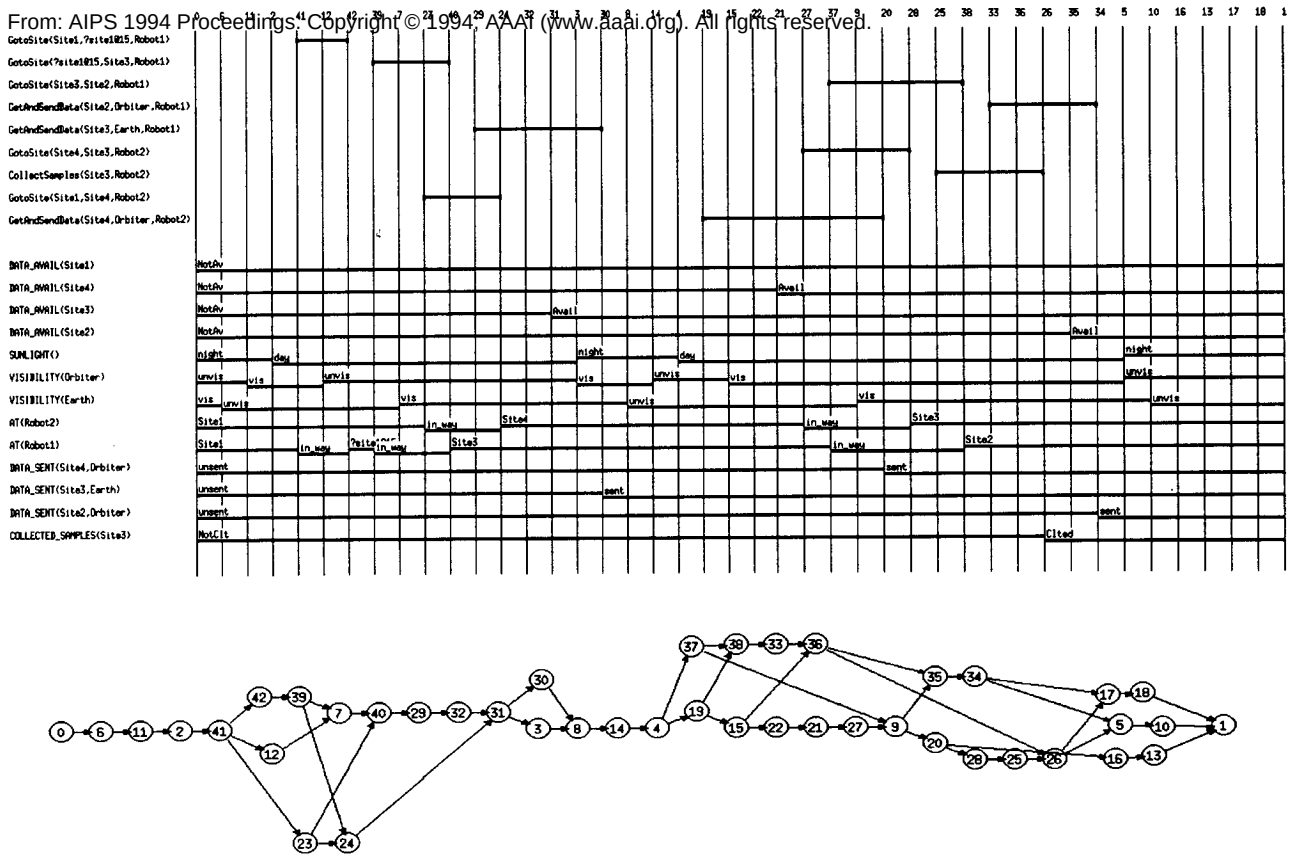

Figure 4: *Plan for two planetary exploration robots*

of tasks. Indeed, the procedure corresponds to a look-ahead where the conflicts between tasks are not considered. Hence, the more independent the tasks are, the deeper the search can be developed, and the better the estimate will be.

To perform this look-ahead, a particular temporal propagation is tentatively computed along the decomposition tree. Basically, it refers to the time-map of the root node.

Experiments and Conclusion

This paper described a representation and algorithms for a temporal planner. The representation of IxTeT relies on a reified formalism, which naturally splits the management of temporal constraints into a temporal network from the management of the other formula. Similarly, instantiation constraints are handled separately in a "variable binding network". An ontology on domain relations allows to specify different types of expected events, controllable or not by the planner. A hierarchy of tasks increases the expressiveness of the representation. A compilation procedure checks the consistency of planning operators, and prepares the work of the planner. On the control aspects, the proposed A_ε algorithm performs a trade-off between the quality of a solution and the complexity of the search. It relies on causal-links to avoid redundancy and on least commitment to reduce backtracking. Specific procedures for goal decomposition and conflict resolution have been defined. A resource management module handling, in a separate network, various type of resources required by tasks and their possible conflicts, has been similarly added to IxTeT [13].

A first version of IxTeT –implemented in CommonLisp– has been used for the task-level control of a mobile robot HILARE. A demonstration has been shown Advanced Mobile Robot EUREKA project. A new version of IxTeT in C++ has also been applied to a planetary exploration project. Here, visibility constraints between the rover and an orbiter, sun-light, connection with the earth and energy constraints require the management of time.

Let us illustrate the performance of IxTeT through a complex plan generated for this application. There are 4 exploration sites on Mars S1 to S4, connected along a circle S1-S2-S3-S4-S1. Two identical robots R1 and R2 are both initially located in S1. They are required to get detailed views of S2 and S4, to send them to an orbiter, and to collect samples from S3. Because of energy constraints, robots travel only by day. Image

acquisition requires also daylight. Data transmission needs the visibility with the target, orbiter or earth. Collecting samples requires daylight and earth visibility, a detailed view of the site should be sent to earth before starting this task. There are four elementary tasks: going to site, collecting samples, getting data, and sending data, each requires one third of a day. There is one compound task: get and send data. Two days are allocated for achieving this mission. Visibility windows with earth have the same duration as a Mars day and they start at mid-day. The orbiter remains visible during 2/3rd of a day (see fig. 4; time-points 11,12,3,14,15 and 5).

These constraints forbid a simple plan where each robot goes to a site S2 or S4 in day 1 (interval [2,3]) and one of them goes to S3 and collects samples in day 2 (interval [4,5]). IxTeT finds the complex plan given in fig 4. Here robot R1 goes to S3 through a free variable ?S@15 (to be instantiated as S2 or S4 at execution time only). Path S1-?S@15 takes place between time-point [41,42], the next path to S3 between [39,40]. R1 acquires a view of S3 [29,32] and sends it by night to the earth [31,30]. During that time robot R2 goes from S1 to S4 [23,24]. The next day R1 goes from S3 to S2 [37,38], acquires and sends data of S2 to the orbiter[33,36,35,34], while R2 acquires data of S4 [19,22], travels to S3 [27,28] while sending data of S4 to the orbiter [21,20] at the same time, and then collects samples on S3 [25,26].

To find this plan, the search develops 5319 nodes, and backtracks 1278 times. It took 25 minutes on a sun sparc station 10. For less constrained situations, the simpler plan given above develops 200 nodes, backtracks 25 times, and takes 1 min.

Several extensions are currently being studied. Domain constraints, static or temporal formula, should be added to the representation. They will permit partially specified initial situations. They should also enable to complete, at preprocessing time, planning operators through deduced effects (eventually conditional) in order to have a partial but computable solution to the ramification problem. At the control level, we are working on a comparison and a characterization of alternative heuristics and domain specific knowledge. There is finally the need for a programming environment to help the user specify complex tasks that would be efficient from the planning point of view.

References

[1] J.F. Allen, H.A. Kautz, R.N. Pelavin, and J.D. Tenenberg. *Reasoning about Plans.* Morgan Kaufmann Pub., 1991.

[2] S. Biundo. Present-Day Deductive Planning. In *2nd EWSP. Vadstena(Sweden)*, December 1993.

[3] D. Chapman. Planning for conjunctive goals. *Art. Int.*, 32:333–377, 1987.

[4] S.J.S. Cranefield. A Logical Framework for Practical Planning. In *10th ECAI*, pages 633–637, 1992.

[5] K. Currie and A. Tate. O-plan: the open planning architecture. *Art. Int.*, 52:49–86, 1991.

[6] T. Dean, R.J. FIRBY, and D. Miller. Hierarchical Planning Involving Deadlines, Travel Time and Resources. In *Comput. Intellig.*, pages 381–398, 1988.

[7] T. Dean and D. McDermott. Temporal Database Management. *Art. Int.*, 32:1–55, 1987.

[8] M. Ghallab and A. Mounir Alaoui. Managing Efficiently Temporal Relations Through Indexed Spanning Trees. In *11th IJCAI*, 1989.

[9] M. Ghallab and D.G. Allard. A_ϵ: an efficient near admissible heuristic search algorithm. In *8th IJCAI*, 1983.

[10] J. Hertzberg and E. Rutten. Temporal Planner = Nonlinear Planner + Time Map Manager. *AI-Communications*, 6(1):18–26, March 1993.

[11] S. Kambhampati. On the utility of systematicity: understanding tradeoffs between redundancy and commitment in partial-ordering planning. In *Proceedings Spring Symposium AAAI*, 1993.

[12] C. Knoblock, J. Tenenberg, and Q. Yang. Characterizing Abstraction Hierarchies for Planning. In *9th AAAI*, 1991.

[13] P. Laborie. Planifier avec des contraintes de ressources (in french). Technical report, LAAS-CNRS, Toulouse (France), 1994.

[14] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings AAAI*, 1991.

[15] Y. Shoham. *Reasoning About Change.* The MIT Press, Cambridge, MA, 1988.

[16] E.P.K. Tsang. TLP: a Temporal Planner. In *Advances in Artificial Intelligence.* Hallam & Mellish ed., 1987.

[17] S.A. Vere. Planning in time: Windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(3), May 1983.

[18] M. Vilain and H. Kautz. Constraint propagation algorithms for temporal reasoning. In *5th AAAI*, 1986.

[19] D. E. Wilkins. *Practical Planning.* Morgan Kaufmann, San-Mateo, CA, 1988.