Hierarchisation of the Search Space in Temporal Planning

Frédérick Garcia
garcia@toulouse.inra.fr
Station de Biométrie et d'Intelligence Artificielle
INRA, BP 27
31326 Castanet-Tolosan cedex, France

Philippe Laborie
laborie@laas.fr
LAAS/CNRS
7, Avenue du Colonel Roche
31077 Toulouse, France

Abstract. Hierarchical problem solving with abstraction is a widely adopted strategy to explore very large search trees. In this paper we describe the automatic generation of abstraction hierarchies for the IxTeT planner. Two points are mainly developped. First, we extend the classical STRIPS-like abstraction formalism to the IxTeT representation, which allows the use of multi-valued variables, resources and temporal constraints. Second, we describe a new way of dynamically managing abstraction hierarchies during planning. In our approach, the definition of the abstraction levels depends not only on the planning problem, but also on the current partial plan in the search. This least-commited hierarchy has been implemented on the IxTeT planner and its interest in terms of reduction of backtracking and global efficiency has been clearly established on several realistic domains.

1 Introduction

Hierarchical problem solving is a widely adopted strategy to explore very large search trees; in planning, this idea dates back to ABSTRIPS [17]. Since then, and especially in these latter years, many results have shown how hierarchisation could help saving a lot of useless search efforts. The majority of these results were obtained under a STRIPS-like formalism but they have rarely been extended to a richest one and confronted with realistic problems. We show in this paper that this extension is achievable while maintaining the properties established for the STRIPS-like formalism. Furthermore, we describe a new way to make a trade-off between hierarchisation and least-commitment thanks to a dynamic instantiation of the hierarchy during planning. This approach is easily generalizable to every least-commitment planner.

Our approach has been implemented on IxTeT, a temporal planner with a great expressiveness of representation well suited to solve realistic problems [5, 7, 13].

Section 2 describes the IxTeT representation of operators and problem and the main principles used in the basic search procedure. Section 3 summarises the state of art in hierarchical planning. The IxTeT hierarchical approach is discussed in section 4 and some experimental results are given in section 5.

2 Representation and Basic Control

2.1 Description of the World

For algorithmic complexity reasons, the IxTeT *time-map manager* [5, 8] relies on time-points as the elementary primitives. Time-points are seen as symbolic variables on which temporal constraints can be posted. We handle both *symbolic constraints* (precedence, simultaneity) and *numeric constraints* expressed as a bounded interval $[I^-, I^+]$ on the temporal distance between time-points. The *time-map manager* propagates these constraints to ensure the global consistency of the network and answers queries about the relative position of time-points.

Atemporal variables management is achieved through a *variable constraint manager*. We consider variables ranging over finite sets and propagate *domain restriction*, *equality* and *unequality constraints*. Constraint propagation is achieved through classical CSP techniques [12].

Properties of the world are described by a set of **multi-valued state attributes** and a set of **resource attributes**.

Each state attribute is a k-ary mapping, from some finite domains into a finite range, called the **value** of the attribute. Domains and their corresponding possible values are explicitly declared. e.g. the position of a robot:

```
attribute position(?robot){
  ?robot in {robot1, robot2};
  ?value in {place1, place2, corridor};
}
```

A more complete description of the IxTeT formalism concerning state attributes can be found in [7].

We define a resource as *any substance or set of objects whose cost or availability induces constraints on the actions that use them.*

A resource can be a single item (with a unit capacity, we speak then of *unsharable resource*) or an aggregate resource that can be shared simultaneously between different actions seeing that its maximal capacity is not exceeded. Resources are gathered together into *resource types*: two resources belong to the same type if they can be indifferently used by the actions.

```
resource robots(?robot){
  ?robot in {robot1, robot2};
  capacity = 1;
}
```

```
resource power(){
  capacity = 100;
}
```

The representation and management of resources are further described in [13].

The names of the state attributes (e.g. `position`) or of the resource attributes (e.g. `power`, `robots`) will be indifferently called **attribute symbols**.

2.2 Description of Change

IxTeT is based on a reified logic formalism [18] where state attributes are temporally qualified by the predicates *hold* and *event* and resource attributes by the predicates *use*, *consume* and *produce*.

- an assertion $hold(att(x_1, ...) : v, (t_1, t_2))$ asserts the persistence of the value of state attribute $att(x_1, ...)$ to v for each t: $t1 < t < t2$. *Assertions* allow the expression of operator conditions as well as causal links.

- $event(att(x_1, ...) : (v_1, v_2), t)$ states that an instantaneous change of value of $att(x_1, ...)$ from $v1$ to $v2$ occured at time t. *Events* allow the description of state change in operators as well as of expected changes of the world in the initial problem description.

In the same way, resource availability profiles and the use of resource by the different operators are described by means of three predicates:

- $use(typ(r) : q, (t_1, t_2))$ represents a borrowing of an integer quantity q of resource r of type typ between time-points t_1 and t_2.

- $consume(typ(r) : q, t)$ states that a quantity q of resource r will be consumed at time t, i.e. that the availability of r will decrease after t.

- $produce(typ(r) : q, t)$ represents a production of resource at time t.

Assertions, events and resource usage are indifferently called **temporal propositions**.

From our reified logic representation, a *hierarchy* of operators is defined, called **tasks**: a task may refer to other sub-tasks, specified with the same syntax. This feature enables the expression of parallelism and combined actions in the description of a task. This representation do not allow recursion, and is only a hierarchy of *description*, meaning that it provides only facilities for user specification. **It is not, for the moment, directly connected to the hierarchy of control discussed in this paper.**

More precisely, a task is a temporal structure composed of:

- a set of sub-tasks;

- a set of events describing the changes of the world induced by the task;

- a set of assertions on state attributes to express the required conditions or some causal links between task events;

- a set of resource usages; and

- a set of temporal and instantiation constraints binding the different time-points and variables of the task.

e.g., in the description of the room finishing problem, as stated in [15], that consists in planning the finishing of a room for three workers (a plumber, an electrician and a paper hanger), we can declare the task consisting in cutting off the water of the room in the following way:

```
task cutoff_water(?worker) (start,end){
  ?worker in {plumber};
  variable ?wall;
  use(worker(?worker):1,    (start,end));
  hold(water_on_wall(?wall):on, (start,end));
  hold(running_water():on, (start,end));
  event(running_water():(on,off), end);
  (end - start) in [00:05:00,00:05:00];}
```

The initial plan $\mathcal{P}_{init}$ is a particular task that describes the problem scenario that is:

- the initial values for the set of instantiated state attributes;
- the expected changes on some contingent state attributes that will not be controlled by the planner;
- the expected availability profile of the resources; and
- the goals that must be achieved.

IxTeT allows a total flexibility on the expression of temporal constraints between these elements. Here is a part of the initial plan for the room finishing problem:

```
task room_finishing_pb() (start,end){
  // initial situation
  explained event(running_water():(?,off),start);
  explained event(elec_conn(w1,w2):(?,off),start);
  explained event(papered(w1):(?,no), start); //...
  // goals
  hold(papered(w1):yes, (t_goal,end)); //...
  start <= t_goal <= end;
  (end - start) in [00:00:00, 24:00:00]}
```

2.3 The Basic Search Procedure

The IxTeT planner explores a search tree of partial plans whose root is the initial plan and branches represent some tasks or constraints insertion on the current plan in order to solve a **flaw**. The search relies on an extension of the SNLP procedure [14] to our temporal formalism. In our representation, causal links are naturally expressed by means of assertions.

We distinguish three kinds of flaws:

- *pending subgoals* are those events or assertions that have not yet been established;
- *threats* are event or assertions that can threaten a protection;
- *resource conflicts* are a set of resource usages (produce, use or consume) that could be conflicting in some instantiation of the current plan.

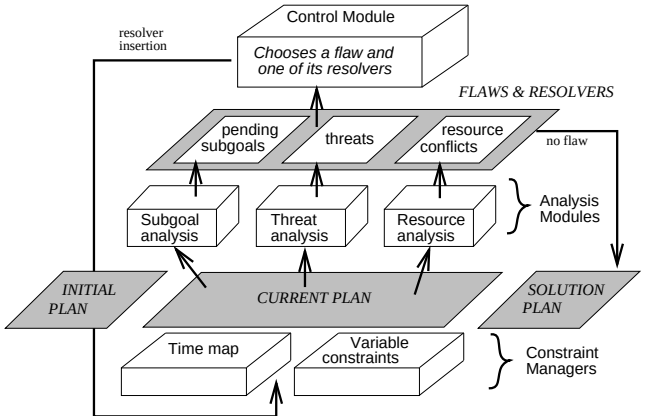
During the search, in a current partial plan, pending subgoals, protection threats and resource conflicts are detected by three *flaw analysis modules* (c.f. fig. 1).

For each flaw, a minimal disjunction of **resolvers** is computed (see fig 1 for a typology of resolvers for a given flaw).

A *control module* chooses the order of flaw resolution independently of the origin of the flaw (pending sub-goal, threat or resource conflict) according to an opportunistic least-commitment strategy based on an *estimation* of the commitment induced by each resolver.

Let ϕ a flaw and $\{\rho_1, ..., \rho_k\}$ the minimal set of its resolvers. The **commitment** of posting resolver ρ_i on the current partial plan $\mathcal{P}$ is an estimation[1] of the ratio of

[1] this estimation, for the different types of resolvers is further described in [13]



Flaws / Resolvers	pending subgoals	threats	resource conflicts
temporal constraints		promotion demotion	scheduling constraints
variable constraints		separation fusion	allocation constraints
causal links	causal–link		
task insertion	new task		resource production

Figure 1: *the IXTET Planner Architecture and the Typology of Flaws*

instantiation of $\mathcal{P}$ eliminated by the posting of ρ_i. For a given selected flaw, this measure allows to choose first to post the least-commited resolvers.

We define the **opportunity** $K(\phi)$ of solving a flaw ϕ as an estimation of the easiness to make a choice between its resolvers:

$$\frac{1}{K(\phi)} = \sum_{i=1}^{k} \frac{1}{1 + commit(\mathcal{P}, \rho_i) - commit(\mathcal{P}, \rho_{min})}$$

Each node of the global search tree is dedicated to the resolution of the most opportunistic remaining flaw in the partial plan. A child node is created for each resolver of this flaw. The global non-deterministic algorithm is presented below.

Algorithm $plan(\mathcal{P})$

1. **termination:** if $\mathcal{P}$ does not contain any flaw: return P
2. **analysis:** call the three flaw analysis modules:
   $FLAWS = pending\_subgoals(\mathcal{P}) \cup threats(\mathcal{P}) \cup resource\_conflicts(\mathcal{P})$
3. **flaw selection:** select the most opportunistic flaw $\Phi \in FLAWS$;
4. **choice of a resolver:** choose least-commited resolver $\rho \in resolvers(\Phi)$;
5. **recursive invocation:** $plan(insert(\mathcal{P}, \rho))$

The search tree is controlled by a near-admissible A_ϵ algorithm [6] which provides a trade-off between the efficiency of the search and the quality of the found solution in terms of flexibility. The estimate f of a partial plan $\mathcal{P}$ at a given node is a combination of g, the commitment along the path leading from $\mathcal{P}_{init}$ to $\mathcal{P}$, and h, a combination of the minimal commitment for each remaining flaw ϕ in $\mathcal{P}$.

2.4 Expressivity of the Representation and Limitations of the Basic Control

The main originality of the IXTET planner is its rich formalism (explicit representation of time and finite domain variables, task and problem formalism, resource management). Unlike many planners relying on a STRIPS-like formalism, IXTET allows a representation of realistic domains (c.f. §5) that requires an explicit management of time and resources.

To support efficiently this rich formalism while ensuring the completeness of the search, we must be able, for each node of the search tree, to choose quickly the most

promising flaw to be solved. The basic control procedure described above shows two important drawbacks from this point of view:

1. as, for each nodes, it explores the whole set of flaws to make a choice, the procedure is rather slow; and

2. the choice criterion ($K(\phi)$) is very local whereas, sometimes, a more global strategy should prevail.

We will see in the following sections how the hierarchisation of the search can help solving these problems.

3 Abstraction hierarchies

The goal selection problem encountered in classical backward planning is a part of the search control strategy which has to be defined in all planning algorithms. Recently, there has been a lot of works on the formalisation of abstraction hierarchies in linear and nonlinear planning [20, 11, 10], with the purpose of defining an intelligent goal selection procedure. Many different experimental results [19, 10] have established the fact that this hierarchical approach was able to save easily a considerable amount of running time.

The principle of abstraction hierarchies for planning, initially due to Sacerdoti with the ABSTRIPS system [17], and reintroduced in ABTWEAK or PRODIGY [20, 10], is quite simple. A natural number, often called *level*, or *criticality*, is associated to each potential goal/subgoal in the planning domain. Generally, starting with the top highest criticality level, the planning problem is solved at each level i, considering only during this step the goals/subgoals with a criticality greater or equal to i. This approach requires two types of investigations : how to assign abstraction levels to goals/subgoals, and how to exploit these levels within the control of the search.

All the results on abstraction hierarchies in planning have been obtained under the STRIPS assumption for the action representation. In this formalism, an action is represented by an operator α as a pair ($pre(\alpha)$, $effect(\alpha)$), where $pre(\alpha)$ and $effect(\alpha)$ are some lists of literals, positive or negative atomic formulas of a first order language, describing the preconditions and the effects of the action. Goals are represented by literals and are given an abstraction level.

Given a planning domain and a problem to solve, many different abstraction hierarchies can be used. In order to characterise these hierarchies, different criteria have been defined. The two most studied criteria are based respectively on the *Downward Refinement* property [1, 2], and on the *Ordered Monotonicity* property [9], which can be stated unformally as follows :

Property 1
Downward Refinement property (DR): *Every abstract plan can be refined to a concrete solution, given that a concrete solution exists.*
Ordered Monotonicity property (OM): *For all abstract plans, every refinement of those plans leaves the literals established in the abstract plan unchanged.*

In these definitions, an abstract plan is a plan solution of the planning problem at a given abstraction level. A plan P_{i-1} is a refinement of an abstract plan P_i if P_{i-1} is

an abstract plan that differs from P_i just by the addition of operators and constraints with the purpose of satisfying preconditions introduced at level $i-1$.

The DR property criterion is clearly the most interesting property that a hierarchy can satisfy. It reduces exponentially the search cost. But is is a very strong criterion, and thus can only be guaranteed in few restricted cases. More appealing in practice is the OM property. When it holds, a lot of sources of backtracks are eliminated since no possible interactions across levels are allowed. In this case, the search efficiency can be very improved. Furthermore, it has been shown that it was quite simple to define an automatically ordered monotonicity abstraction hierarchies, making attractive the use of this method for planning [10].

In order to produce automatically an abstraction hierarchy, Knoblock has proposed different sufficient conditions to guarantee the OM property. These conditions correspond to a set of constraints on the literal levels that are derived from the syntactic description of the operators and of the problem:

Property 2 (sufficient conditions for the OM property)
Let O be the set of operators in a domain. $\forall \alpha \in O$, $\forall p \in pre(\alpha)$ and $\forall q, q' \in effect(\alpha)$,

1. $level(q') = level(q)$
2. $level(p) \leq level(q)$

The principle of these conditions is simple : they require the literals that could possibly be changed when achieving some other literals to have a level number lower or equal to those last literals. It guarantees that any operator for achieving directly or indirectly a literal will not add or delete a literal with a higher level number.

From the set of all the previous sufficient conditions obtained from a given planning domain, it is very simple to generate an abstraction hierarchy which satisfies the OM property. One just has to find a level valuation of the literals satisfying the constraints. The solution proposed in the system ALPINE [10] consists in constructing the graph the nodes of which are literals (or sets of literals with equal levels), and where the directed edges represent level constraints between these nodes. Then a topological sort of the graph is applied, which defines a total order on the literals. This total order leads to a decomposition in abstraction levels that satisfies the OM property.

Some extension to this model have been proposed [10, 3]. Fink and Yang have introduced the notion of *forbidding precondition*, which stands for preconditions l of actions α such that once l does not hold, the preconditions of α can never be achieved. Restricting the previous sufficient conditions to non-forbidding preconditions defines sufficient and necessary conditions to satisfy the OM property. The problem is that it seems very difficult to define an efficient algorithm to find all the forbidding conditions.

It can also be interesting to take into account the problem to solve for generating an abstraction hierarchy. The expected advantage is that the hierarchies will be less unnecessary constrained, allowing more freedom to the search control. The solution proposed by Knoblock consists in considering as possible effects in the previous conditions only the ones *relevant* for the goal.

Another possible extension consists in distinguishing between the *primary* and the *secondary* effects of an operator. Classically, the primary effects specify the real purpose of the corresponding action, and the secondary effects are side effects of this action;

During the planning process, operators are selected to achieve a goal only by considering primary effects. In the sufficient conditions for the OM property, only primary effects are retained.

The last extension proposed concerns the use of operator templates. Indeed, in most of the planning systems, operator templates are used instead of instantiated operators. The solution proposed by Knoblock is to associate a type with each instantiated or uninstantiated literal present in the operator template description. Theses types are determined by both the predicate and the argument type, each constant and variable having an associated type.

4 The I$_X$T$_E$T Least-Commitment Hierarchy

For the reasons presented in the two previous sections, the use of an abstraction-hierarchy based control in the I$_X$T$_E$T planning system is something very appealing. We propose in this section a new principle for automatically generating a least-commited abstraction hierarchy, well suited for the I$_X$T$_E$T formalism.

4.1 Abstraction hierarchy for I$_X$T$_E$T

We have seen in section 2.3 that the control of the search in the I$_X$T$_E$T planning system consists in choosing in the current partial plan a **flaw** to be solved.

For this flaw, a minimal set of resolvers is computed from which we can generate the successor nodes of the current partial plan. Thus, the generalisation of the previously presented works on hierarchisation relying on a STRIPS like formalism leads us to assign abstraction levels to the different flaws. In I$_X$T$_E$T, flaws can be of three different kinds : pending subgoals, threats and resource conflicts. For each flaw instance, one unique *attribute symbol* is present. From this observation and the control facilities it implies, we have chosen in a first approach to characterise a flaw level by its attribute symbol:

Hypothesis 1 (one level per attribute) *: Abstraction levels are sets of flaws. Two flaws with the same attribute symbol are necessarily in the same level.*

At the control level, the management of an abstraction hierarchy consists in solving the flaws of the different levels, from the more abstract level to the ground level. We have chosen to search for hierarchies satisfying the following OM like property :

Property 3 (Ordered Monotonicity property for I$_X$T$_E$T) *: For all possible current partial plans, every refinement of those plans only creates new flaws belonging to the current or next abstraction levels.*

Planning with an abstraction hierarchy verifying this property implies that the only reason to backtrack from the current partial plan is the absence of resolvers for a given flaw of the current level or the presence of inconsistent constraints. The efficiency of this approach has been experimentaly confirmed in many contexts.

Despite the simplicity of this criterion, different types of hierarchisation are possible. The majority of planners using abstraction hierarchy (ABSTRIPS[17], ABTWEAK [20], PRODIGY[10]) imposes a pre-established total-ordered hierarchy for planning.

More precisely, the hierarchy they use is defined as a pre-established partition of the flaw set, each subset of which defines an abstraction level. The way they define such a decomposition consists in, first, finding a partial-order hierarchy that satisfies the OM property (cf. property 2) and then, to order it according to an off-line strategy [10, 2]. This approach raises an important problem: as this strategy is static and decorrelated from the planning process, it leads to an excessive commitment that may result in additional backtracking across abstraction levels.

In consequence, we propose a different solution; in the IxTeT planner, we define directly the hierarchy within the planning process. The different levels of the global hierarchy are built on-line, according to the evolution of the search. If L represents the current abstraction level, that is the set of all the flaws possibly currently in process, the main idea of our approach is not to wait for the absence of L's flaws in the current partial plan before changing abstraction level from L to the next level L', as it is the case in classical abstraction hierarchies.

4.2 A least-commitment approach

An important issue is how to construct automatically such an IxTeT abstraction hierarchy. Similarly to the ALPINE system, our approach relies on the definition of sufficient conditions from the syntactic description of the domain tasks. These conditions express constraints on the possible abstraction levels and abstraction hierarchies. They are represented by a partial order on the set of attribute symbols.

In the following definition we assume that all the subtasks of a given task have been already recursively developed, that is a task only contains sets of events and assertions, resources uses and temporal and instantiation constraints. We denote by att_p the attribute symbol present in a temporal proposition (assertion, event or resource usage) p and att_ϕ the attribute symbol of a flaw ϕ.

Definition 1 (ordering constraints $\prec$) *: For all tasks T of the domain. If e is an* event *or a* resource production *of T and if p is a temporal proposition of T, then $att_p \prec att_e$*

The presence of the constraint $att \prec att'$ implies that during the planning process, it should not be allowed to resolve flaws with attribute symbol att in a more abstract level than flaws with attribute symbol att'. Then, from these constraints $\prec$, one can define a relation $<$ between sets of $\prec$-equivalent attributes (we identify $\prec$ and its transitive closure):

Definition 2 ($<$) *: Let $\overset{\circ}{att} = \{att' \mid att \prec att' \wedge att' \prec att\}$. Then $\overset{\circ}{att} < \overset{\circ}{att}'$ if and only if $\overset{\circ}{att} \neq \overset{\circ}{att}'$ and $att \prec att'$.*

This partial-order $<$ can be represented by an acyclic constraint graph $\mathcal{G}$ the nodes of which are sets of $\prec$-equivalent attribute symbols, and oriented edges represent constraints on the abstraction levels.

To illustrate these definitions, scanning the task `cutoff_water` described in section 2 imposes the constraints `worker` $\prec$ `running_water` and `water_on_wall` $\prec$ `running_water`. The figure 2 shows the graph $\mathcal{G}$ automatically generated from the room finishing domain;

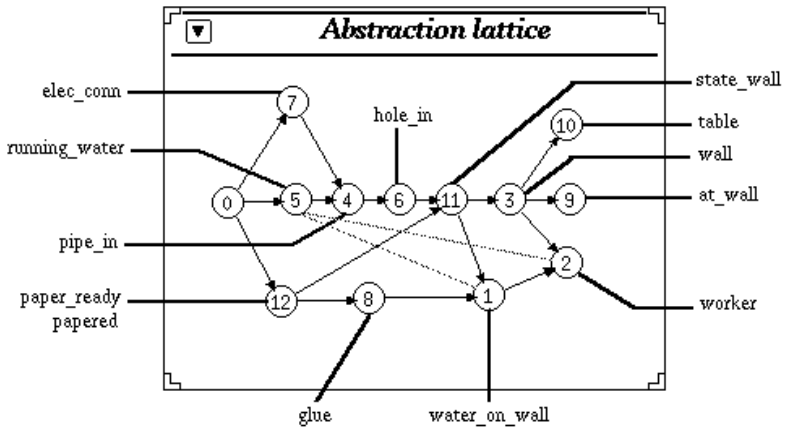


Figure 2: *graph $\mathcal{G}$ for the room finishing domain*

Once ordering constraints on the attribute levels have been generated, we have to explicit an abstraction hierarchy that satisfies the IxTeT OM property and a control procedure that uses it.

In the IxTeT planner, we manage directly the partial-ordered hierarchy within the planning process. With this aim, we introduce the notion of **abstraction state**:

Definition 3 (abstraction state) *: An abstraction state is a pair (S, C) of attribute symbols, where $S = \{att_{s1}, ..., att_{sn}\}$ represents the set of attributes that have been completely solved so far, and $C = \{att_{c1}, ..., att_{cm}\}$ is the set of attribute symbols that are currently in process.*

An abstraction state is associated to each node of the global search tree: only the flaws corresponding to those attributes in C are examinated. The C set of the current abstraction state characterizes the current abstraction level L:

$$L = \{\phi \mid att_\phi \in C\}$$

Hence, to define an abstraction hierarchy is equivalent to define the way the abstraction states are transformed along the nodes of the search tree.

The initial abstraction state of the root node is set to (S_0, C_0), with $S_0 = \emptyset$, and $C_0 = \{att \mid \{att' \mid \overset{\circ}{att} < \overset{\circ}{att}'\} = \emptyset\}$. Then let us assume that during the process of the level L_0, the set $\overset{\circ}{att}$ of $\prec$-equivalent attribute symbols have been treated, with $att \in C_0$. Thus the current partial plan does not contain any flaw with some attribute symbols in $\overset{\circ}{att}$, and we can be sure that these attributes won't come up again. Indeed, the definition of C_0 prevents us from the existence of a flaws ϕ in L_0 the treatment of which would lead to the creation of a new flaw ϕ', with $att_{\phi'} \in \overset{\circ}{att}$.

Since these $\prec$-equivalent attribute symbols of $\overset{\circ}{att}$ can no more appear, they can be deleted from the C set of the current abstraction state. The classical hierarchical approach would consist in waiting for the set C to be empty before calculating a new set C' and a new abstraction level L'. In fact, this is not necessary and we can immediately add to C some new attribute symboles to treat, without violating the $\prec$ constraints that insure the OM property.

Hence, the general update rule of the current abstraction state (C, S) in a search node is the following:

Definition 4 (abstraction state update) *: When the set $\overset{\circ}{att}_{solved}$ of $\prec$-equivalent attributes is completely solved, $\overset{\circ}{att}_{solved}$ is removed from C, added to S and C is recomputed from the new set of solved attributes S.*

$$S' = S \cup \overset{\circ}{att}_{solved}$$
$$C' = C - \overset{\circ}{att}_{solved} \cup \{att_{new} \mid \overset{\circ}{att}_{new} < \overset{\circ}{att}_{solved} \wedge \{att \mid \overset{\circ}{att}_{new} < \overset{\circ}{att}\} \subset S'\}$$

An instance of abstraction state update is shown on the figure 3. We supposes attributes `paper_ready` and `paper_on` have already been solved in the initial abstraction state (S, C).

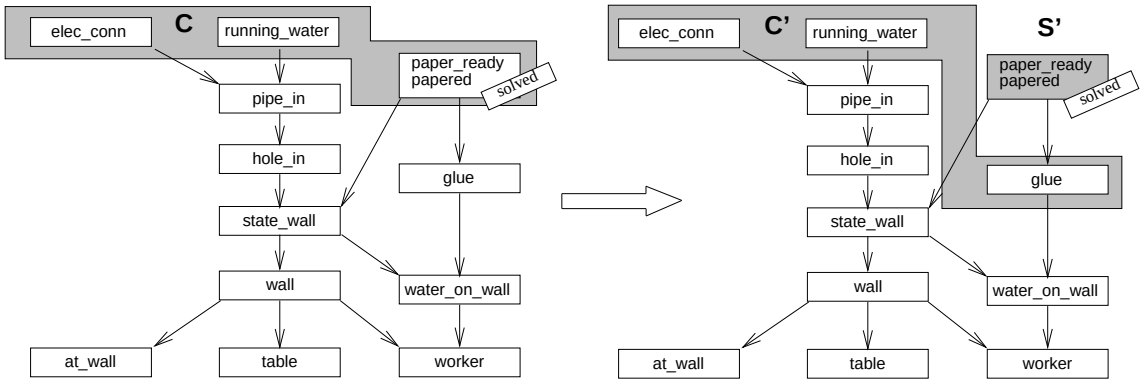


Figure 3: *two consecutive abstraction states while solving the room finishing problem*

Then, like in the classical hierarchical approach, the following theorem holds [4]

Theorem 1 *: The $I_X T_E T$ least-commitment hierarchies are ordered monotonic abstraction hierarchies: for all possible current partial plans, every refinement of those plans only creates new flaws belonging to the current or next abstraction levels.*

The main difference with the classical abstraction hierarchy definitions is that the levels we generate do not form a partition of the flaw set: two successive levels generaly intersect. The advantage of this approach is that the control is more opportunistic, just dealing with the constraints present in the graph $\mathcal{G}$, and no more. It is thus possible to develop in priority the most opportunistic parallel branches of the graph $\mathcal{G}$ and wait for a more informed partial plan to solve the less opportunistic ones.

Let us define the abstraction depth $d_{\mathcal{G}}$ as the longest path in the graph $\mathcal{G}$ between two nodes, and $p_{\mathcal{G}}$ the maximal degree of parallelism[2] in $\mathcal{G}$. We denote by $n_{\mathcal{G}}$ the number of nodes of $\mathcal{G}$. The following results characterize least-commitment abstraction hierarchies, independently of the specific problem which has to be solved [4]

Property 4 *: For each abstraction state (S, C), we have $S \cap C = \emptyset$. C is the reunion of $\overset{\circ}{att}$ sets which are $<$-independant; Their number is up-bounded by $p_{\mathcal{G}}$. The number l of different levels in a least-commitment hierarchy along the path leading from $\mathcal{P}_{init}$ to any solution node $\mathcal{P}$ is such that $d_{\mathcal{G}} \leq l \leq n_{\mathcal{G}}$.*

[2] this degree is defined as the size of the maximal independant set for the partial-order $<$

5 Implementation, Results and Limitations

5.1 Automatic Abstraction Generation

Our hierarchical approach of planning has been implemented on the IXTET planner. An *Abstraction Generation module* allows the automatic generation of the least-commited graph $\mathcal{G}$ described above by posting only the sufficient ordering conditions to satisfy the OM property. Eventually, the user can interactively add any additional constraint he wishes to furtherly structure the hierarchy.

The ordering constraints $\prec$ are represented as an IXTET lattice [5] where each node represents a set of attribute symbols that must necessarily be handled at the same abstraction level. This lattice is initialised with as many unconstrained nodes as there are different attribute symbols in the domain. The tasks that are relevant for the given problem are, then, recursively scanned and the sufficient conditions to satisfy the OM property are posted on the lattice. Notice that we have implemented on the IXTET planner the notion of primary/secondary effects of a task: a task can be inserted in a partial plan only because of its primary effects and thus, only these effects need to be considered while generating the abstraction hierarchy. Whenever posting a constraint induces a cycle in the oriented graph, the nodes of this cycle are merged into a single one. The efficiency of the constraint propagation and request answering on the IXTET lattice (both of them are linear in average) allows a global complexity of the abstraction generation process in $O(m.n.s.e)$ if m is the number of tasks, n the number of attribute symbols, s the average size of the task in terms of temporal propositions and e the average number of primary effects in a task (generally, $e = 1$ or 2). An example of automatically generated constraint graph is given in fig 2.

5.2 Abstraction Management

As stated above, to each node of the global search tree corresponds a local *abstraction state* (S, C). Whenever an attribute symbol att in C is completely solved, a check is done to see if the set $\overset{\circ}{att}$ of attribute symbols $\prec$-equivalent in the abstraction lattice are also solved; in this case, the abstraction state is modified according to the definition 4.

Fig 3 shows an example of two consecutive abstraction states in the process of planning for the room finishing problem.

At a given node of the global search tree, the overhead of such a procedure is in $O(n.p_{\mathcal{G}})$ if the abstraction level is modified, $O(n)$ otherwise. This overhead is negligible when compared to the complexity of the analyse of the current partial plan.

We have encoded the room finishing domain with 9 state attributes (describing the states of the different water and wire pipes, paper sheets and walls), 4 types of resource (workers, walls, table, glue) and 12 tasks. The problem states that all the walls must be papered, that a given wall $w1$ must have running water and two walls $w3$ and $w4$ must be electrically connected. IXTET, with the constraint graph $\mathcal{G}$ automatically generated and shown in fig. 2 find a solution involving 21 tasks in about 7s on a station Sparc 10. A part of this solution is given in fig. 4.

Our approach has been validated on several other domains such as the task-level

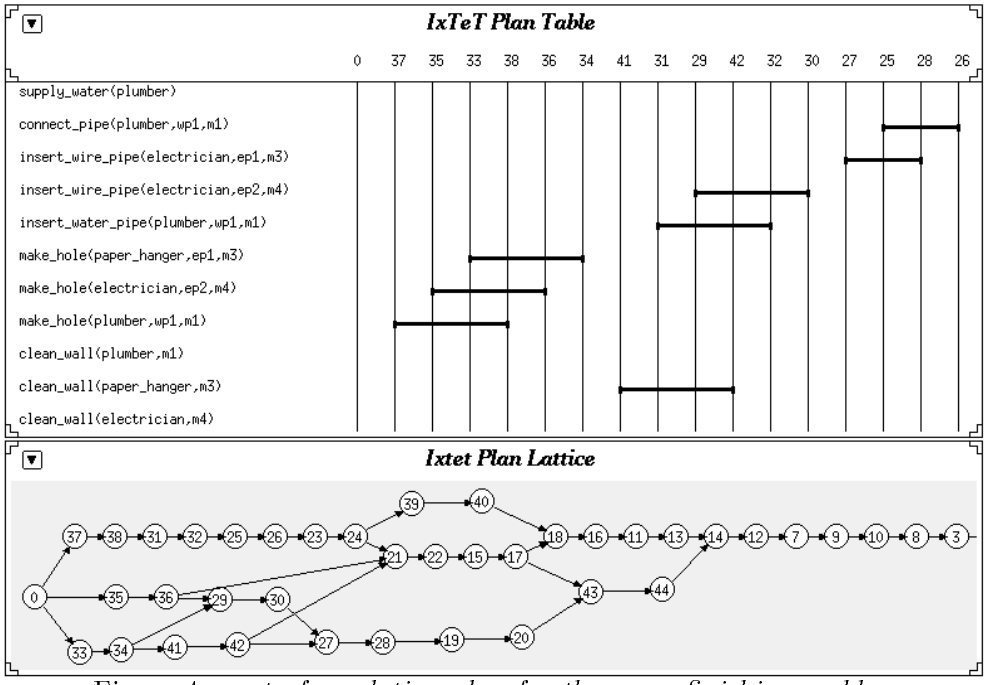


Figure 4: *part of a solution plan for the room finishing problem*

control of a team of robots (Hilares, Iares[3]), the planning of biological experiments in a space lab (Columbus[4]), or the scheduling of equipment compartments integration for a rocket launcher (Ariane[5]). The main results are summarised in table 1. The solution plans to these problems involve about 20-30 tasks, except for Ariane 4 which is a pure scheduling problem for about a hundred tasks. The gain factor is the ratio of the running time without any abstraction by the running time while using our least-commitment abstraction approach.

Domain		gain factor	depth $d_{\mathcal{G}}$	$n_{\mathcal{G}}/p_{\mathcal{G}}$
Hilares		1.4	2	1.5
Iares		2.7	3	1.5
Columbus	problem 1	2.3	4	1.7
	problem 2	1.7	4	1.7
Room finishing		5.0	6	4
Ariane		5.0	6	6

Table 1: *global hierarchy gains*

	off-line ordered hierarchy:	dynamic hierarchy:
Nodes	305	59
Backtracks	14	0
Running time	14.1 s	2.8 s

Table 2: *least-commitment hierarchy gain*

There are two important consequences of the use of hierarchy in the search control. These results were shown on the IXTET planner but as they rely on our least-commitment strategy, they also stand for the majority of least-commitment planners.

1. As only a sub-set of the flaws are examined, the analyse of the partial plan in a given node is faster than without hierarchy. The average time spent at a node is, in average, proportional to the number of attributes in C, and so, it depends on the topology of graph $\mathcal{G}$. The abstraction depth $d_{\mathcal{G}}$ and the ratio $n_{\mathcal{G}}/p_{\mathcal{G}}$ between the size of the abstraction graph and its maximal degree of parallelism are two

[3]this domain was encoded in IXTET for the IARES project in collaboration with the National Centre for Space Studies (CNES)

[4]this domain was encoded within the framework of the PADRE project in collaboration with the companies IXI and Matra Marconi Space France (MMS)

[5]this problem concerning the European launcher Ariane 4 was submitted us by MMS

factors that allow an estimation of the expected gain. In our examples, the gain factor was ranging from 1.4 (Hilares) to 4.5 (Ariane).

2. The hierarchisation allows a structuring of the search space that leads, generally, to a reduction of the chances of backtracking in the search tree because the most relevant features of the plan are solved first. The least-commitment approach described in this paper allows a better exploitation of this property than the traditional off-line computed hierarchies (table 2 illustrates this effect on a very constrained problem in the Iares domain where our dynamical approach finds a solution without backtracking and about 5 time quicker than a static one).

On the other hand, hierarchisation means that the control will focus on a given abstraction level of the plan and disregard the more concrete levels. As a consequence, the heuristic h measuring the distance from the current node to a solution node will be less informed so that, though the hierarchisation reduces the chances of backtrack, whenever a backtrack occurs, choosing the node where to backtrack in the hierarchy is a difficult problem (we could take into account all the flaws of the current partial plan $\mathcal{P}$ to calculate the heuristic h, but that would lead to lose the important observed gain on the initial analysis of $\mathcal{P}$).

6 Conclusion and Future Work

We have seen, in this paper, how a hierarchy criterion (the OM property), initially formulated on a STRIPS-like formalism could be extended to a richest one where the need for a structuring of the search is even stronger. Our work is based on the I$_{\text{X}}$T$_{\text{E}}$T formalism which supports both time and resource constraints, and allows a great flexibility in the expression of operators.

Following the least-commitment strategy used in I$_{\text{X}}$T$_{\text{E}}$T (and in the majority of present day planners), we develop a dynamical hierarchy depending not only on the planning problem (although we showed in [4] that taking into account the planning problem was not as much interesting within I$_{\text{X}}$T$_{\text{E}}$T than in the STRIPS formalism) but also on the current partial plan in the search. Only those hierarchical constraints necessary to ensure the OM property are considered.

The interest of this hierarchisation of the search, in terms of reduction of backtracking and global efficiency has been clearly established on several realistic domains.

An important question that still needs to be adressed concerns the theoritical proof of the relevance of the OM property and the associated search controls, like the one we presented here. Some interesting results have been already obtained concerning the similar OR property [2], or the variable selection problem for CSP [16], but yet no complete analysis has been proposed.

Finally future work will consist in refining the granularity of our hierarchy to handle partially instantiated attributes and variable constraints, and also to exploit the temporal dimension of I$_{\text{X}}$T$_{\text{E}}$T. That would lead to associate an abstraction level to a given flaw according to the partially instantiated attributes or to its temporal position in the partial plan. These developments should reinforce the fact that current abstraction technics in planning are more related to flaw selection controls than to hierachical problem solving strategies.

References

[1] F. Bacchus and Q. Yang. The downward refinement property. In *Proceedings IJCAI-91*, pages 286–292, 1991.

[2] F. Bacchus and Q. Yang. Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence*, 71:43–100, 1994.

[3] E. Fink and Q. Yang. Forbidding Preconditions and Ordered Abstraction Hierarchies. In *Proceedings Spring Symposium AAAI*, 1993.

[4] F. Garcia and P. Laborie. Hierarchisation for the IxTeT Planner, Extended version. Technical report, LAAS/CNRS, 1995.

[5] M. Ghallab and A. Mounir Alaoui. Managing Efficiently Temporal Relations Through Indexed Spanning Trees. In *Proceedings IJCAI-89*, 1989.

[6] M. Ghallab and D.G. Allard. A_ε: an efficient near admissible heuristic search algorithm. In *Proceedings IJCAI-83*, 1983.

[7] M. Ghallab and H. Laruelle. Representation and Control in Ixtet, a Temporal Planner. In *Proceedings AIPS-94*, pages 61–67, 1994.

[8] M. Ghallab and T. Vidal. Focusing on a Sub-graph for Managing Efficiently Numerical Temporal Constraints. In *Proceedings FLAIRS-95 (to appear)*, 1995.

[9] C. Knoblock. Learning abstraction hierarchies for problem solving. In *Proceedings AAAI-90*, pages 923–928, 1990.

[10] C. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68:243–302, 1994.

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

[12] V. Kumar. Algorithms for constraint satisfaction problems: a survey. Technical Report 91-28, Dpt of Computer Science, University of Minnesota, Minneapolis, 1991.

[13] P. Laborie and M. Ghallab. Planning with Sharable Resource Constraints. In *Proceedings IJCAI-95 (to appear)*, 1995.

[14] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings AAAI-91*, pages 634–639, 1991.

[15] L. Missiaen. *Localized Abductive Planning with the Event Calculus*. PhD thesis, Dept of Computer Science, K.U.Leuven, 1991.

[16] P. W. Purdom. Search rearrangement backtracking and polynomial average time. *Artificial Intelligence*, 21(1,2):117–133, 1983.

[17] E. D. Sacerdoti. Planning in a Hierarchy of Abstraction Spaces. *Artificial Intelligence*, 5:115–135, 1974.

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

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

[20] Q. Yang and J. D. Tenenberg. ABTWEAK : Abstracting a Nonlinear Least Commitment Planner. In *Proceedings AAAI-90*, volume 1, pages 204–209, 1990.