Integrating a temporal planner with a path planner for a mobile robot

Brigitte Lamare, Malik Ghallab
LAAS-CNRS
7 avenue du colonel Roche
31077 Toulouse cedex, FRANCE
lamare@laas.fr malik@laas.fr

Abstract

General planners enable to describe unrestricted classes of planning problems, but have the drawback of being relatively slow when compared to the performance of domain-specific planners within their domains. This slowness proves to be a major handicap when faced with planning and executing in a real world environment. In order to overcome this, we have investigated the idea of using domain-specific planners to assist a general one.
We have tested this concept by integrating the general planner IxTeT with a path-finding planner. In our examples, where there are many movements to be planned, the benefits thus obtained are significant.
In order to achieve the integration of the two planners, we have exploited IxTeT's use of an abstraction hierarchy, which enables us to control the planning system formed by the two planners.

Keywords: domain-specific planner, temporal planner, abstraction.

Introduction

A well-known limitation of general planners is that although they offer the advantage of addressing all sorts of problems, they also suffer a lack of performance when compared to a domain-specific planner.
Many studies have been lead in order to improve the overall performance of general planners. Two different approaches seem to emerge from these studies: either specialising a part of the planning process (e.g. time, conflict or resource management), or using a specialised planning process, for example to resolve certain domain-specific problems.
Some planning systems, like O-PLAN2 and IxTeT, use specific constraint managers for time, variable and resource constraints. In O-PLAN2, these constraint managers are supervised by a constraint associator [9], which mediates between the actual planning part of the system and the constraint managers, and handles eventual conflicts between managers.
Other planning systems use an original approach for the planning process itself. UCP [13] opportunistically interleaves plan-space and state-space (both backward and forward) refinement to benefit from both methods. J. Hertzberg suggests building a planning toolbox, that would allow the user to build exactly the system he would like for a given problem [8].
In [12], the authors use several domain-independent heuristics with PRODIGY, that are changed according to the problem to be solved. In the TLPlan system [3], some domain-dependent information is used to guide the forward-chained search of the planner.
The approach that we have chosen is to use a general purpose planner combined with a domain-specific one. A similar approach is used in CEP [2]. CEP is a system that consists in a general planner which elaborates the main tasks, assisted by a domain-specific planner which refines certain of these tasks. However, we do not wish to use the domain-specific planner to refine tasks already produced by the general planner, but to replace the general planner's whole computation of these specific tasks.
This work has been motivated by a robotics domain. Our purpose is to integrate a general temporal planner on board of a mobile robot; this robot will evolve in a real-world, dynamic environment, and be controled by a supervisor overviewing plan production and execution. On the one hand mobility is a significant task in most problem instances and a general planner devotes a large part of its efforts to it. On the other hand, our robotic system has several domain-specific planners, more efficient for their specific tasks.
We will begin by explaining in more detail in what our approach consists, we will then give a description of the two types of planners that we have linked together. We will explain how the change of planning process is controlled, and will give some of the results obtained using this method, before concluding.

Our approach

The general planner that we use is called IxTeT (IndeXed TimE Table), [5], [7], [6]. It already includes specialist time, variable and resource managers. The expressiveness of IxTeT enables to solve a wide range of rich planning problems.

IxTeT has been used aboard mobile robots on the task-planning level. These robots also have specific planners available: several classified path-planners (itinerary planners, a trajectory planner with sensor feedback primitives), a robotic arm command generator, a camera command generator. However, in planning a mission, IxTeT also uses the constraints of the topological graph associated to the mission. The itineraries computed by IxTeT will then be transformed into paths and sensor based movement strategies. However, IxTeT is not very efficient when it comes to calculating an itinerary. Likewise other general planners, IxTeT's planning process proves to be too costly for this type of problem. Typically, it will take IxTeT about 73s to plan a mission in an environment with 10 rooms (the corresponding graph has 10 nodes and 28 edges), whereas it only takes 2s to plan the same mission without taking into account the robot's movements[1]. As for an itinerary planner, the result would be near instantaneous on the movement part of the problem.

If we can link the general planner with the specialist planner, by getting the two planners to cooperate, we will benefit from the advantages of both systems. Our idea is to sub-contract the domain-specific planner to find a solution to part of IxTeT's planning problem. This is what we have done with IxTeT and an itinerary planner, in such a way that the itinerary planner supplies IxTeT with information to complete its partial plan.

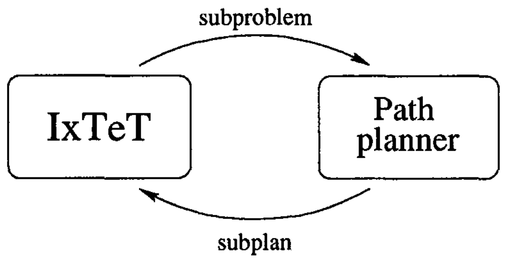


Figure 1: The cooperation

The general idea (figure 1) is to let IxTeT plan normally until it comes across a problem that the domain-specific planner is better suited to manage. When this happens, the domain-specific planner computes a plan from the information that IxTeT has given it. It then feeds back this plan to IxTeT. The latter can then continue with its normal planning process.

[1]calculation times on a Sparc Ultra machine

The general planner

Our study is based on the specificities of the general planner that we have used: IxTeT. Our study is based on the specificities of the general planner that we have used: IxTeT. IxTeT is a temporal planner and searches through a plan-space, guided by an Aϵ-type algorithm. It uses the least commitment approach: each choice is made to constrain the plan the least possible. IxTeT is hierarchical in the sense that the attributes that appear in the problem description are ordered according to their importance in the plan.

Representation and overall control

IxTeT is based on a reified temporal logic formalism [14]. The predicates of this logic express

- the persistence of the value of an attribute over a laps of time declared by an assertion
$$hold(att(x_1,..,x_n) : v, (t_1, t_2))$$
- the change of value of an attribute at a given time-point
$$event(att(x_1,..,x_n) : (v_1, v_2), t)$$

There are also predicates for resource management, which we will not detail here. A description of resource representation and management can be found in [11].

The possible actions of the planning problem are expressed as *tasks*, consisting of a set of events (the effects), a set of assertions (the preconditions or causal links between task events) and a set of constraints between time-points or variables of the task.

The initial state of the world, the goal, and possible events occurring at time-points between the start and end of the plan are all described in a task which is the initial plan P_{init}.

There are three types of flaws to be solved:

- pending subgoals, which are unexplained events or assertions
- threats on protections of the plan

- resource conflicts

The non-deterministic algorithm used by the planner is the following:

```
$PLAN(\mathcal{P})$
  1. Termination: if $\mathcal{P}$ is consistent then
                                  return $\mathcal{P}$
  2. Analyse : calculate the $FLAWS(\mathcal{P})$
  3. Select a flaw : $\Phi \in FLAWS(\mathcal{P})$
  4. Choose a resolver :
    4.1. If $resolvers(\Phi) = \emptyset$ return $failure$
    4.2. Else choose $\rho \in resolvers(\Phi)$
  5. Recursive invocation : return $PLAN(\mathcal{P} \oplus \rho)$
```

$\mathcal{P}$ is the current partial plan and the operator $\oplus$ symbolises the insertion of a resolver in the partial plan.

Abstraction

IxTeT is a hierarchical planner, in the way that its control is based on an Ordered Monotonicity Property similar to the one described in [10].

Ordered Monotonicity Property for IxTeT (OMP). For all possible current plans, each refinement of those plans only creates new flaws belonging to the current or next abstraction levels.

This property orders the planner's search space. The flaws appearing in the partial plan will be processed according to the order of the attribute of each flaw in the hierarchy. Thus all the flaws of an attribute att_i will be resolved before the planner attempts to resolve the flaws of attribute att_j, att_j being of lower lever than att_i. This is an important point for the following explanations about the functioning of the cooperation of the two planners.

IxTeT's different abstraction levels are built in-line, according to the evolution of the search, and to a partial order $<$, defined when the tasks are compiled and that complies with the OMP property.

A partial order $\lhd$ over attributes is defined as follows.

Definition 1
Let att_p be the name of the attribute of a temporal proposition p (event, assertion or resource use).
For each task 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 $att_p \lhd att_e$.
We identify $\lhd$ and its transitive closure.

The order $att_p \lhd att_e$ means that IxTeT will not be able to resolve flaws with attribute symbol att_p before all flaws with attribute symbol att_e have been resolved (providing $att_e \lhd att_p$ is not also true).

Definition 2
Let att and att′ be the names of two attributes.
The abstraction class of attribute att, [att], is the $\lhd$-equivalence class of att, such that

$$[att] = \{att'; att \lhd att' \wedge att \lhd att'\}$$

Given these definitions, we can now give the definition of the partial order $<$ that we mentioned earlier.

Definition 3
Let att and att′ be the names of two attributes, [att] and [att′] their abstraction classes.
$[att] < [att']$ if and only if $[att] \neq [att']$ and $att \lhd att'$.

The abstraction hierarchy built according to these rules can be represented by a graph where the nodes represent abstraction classes and an edge from node j to node i means that $[i] < [j]$, $[i]$ and $[j]$ being the abstraction classes associated to each node (figure 2). IxTeT explores the abstraction graph dynamically during the planning process. If L represents the current abstraction level, that is to say the set of flaws of the partial plan that can currently be analysed, then IxTeT doesn't wait until all flaws of L have been resolved before changing to level L'. This is done by taking into consideration in L' all attributes whose predecessors in the partial order have had all their flaws resolved.

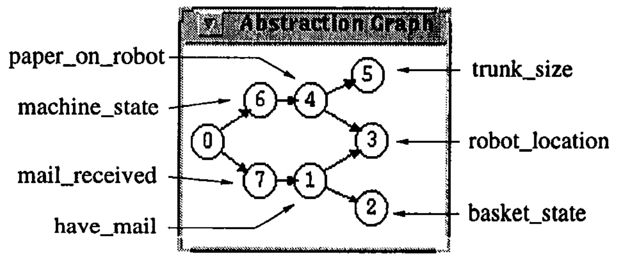


Figure 2: An abstraction graph

A more detailed description of IxTeT's least commitment hierarchy can be found in [4].

The domain-specific planner

To be more precise, the specialised planner only knows how to manage one specific attribute. Its entries are the description of the world, the initial situation, a set of goals to be obtained, and a set of constraints on these goals.
The constraints give a partial order on the goals, thus enabling to express a certain order of achievement of the goals in the resulting plan. If there are solutions to the problem, the planner returns a single solution plan.

We intend to eventually bind the general planner with several different domain-specific planners, but to start with, we have linked IxTeT with an itinerary planner. Its entries are the topological graph of the problem, the starting point in the graph, a set of places (nodes) to be visited, and a partial order in which to visit these places. The topological graph is valuated and its edges are directed.

For example, the topological graph could be that of figure 3, the starting point node $R2$, the places to be visited M, $R3$ and $R4$ and the constraints restricted to "M to be visited before $R4$". The fact that the starting point should be visited before all others is implicit. A resulting solution could be the path $R2$ $R3$ $I2$ M $D4$ $I1$ $R4$.

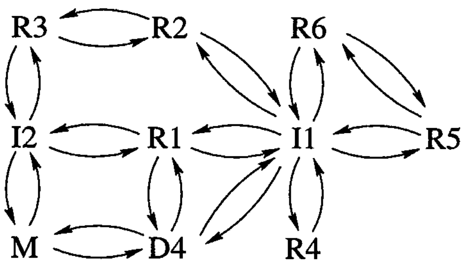


Figure 3: Topological graph

As in the example, the planner may return a solution with places that were not specified in the request, but that have to be in the solution path in order to meet all the requirements, or that are on a less costly path. The algorithm that searches through the topological graph given by the user is based on the Moore-Dijkstra algorithm. It gives one of the shortest paths -in terms of cost- that satisfies the conditions submitted.

Linking the two planners

The principle

The specialised planner is going to be used to solve all flaws concerning a given attribute, which we have called "special attribute". When IxTeT is linked to the itinerary planner the special attribute will be the robot location attribute. IxTeT will consider this attribute as a special one: it will not be dealt with in the ordinary way. All flaws involving the special attribute will be left open for IxTeT: the specific planner will handle them. IxTeT will handle all other attributes as usual. The fact that a certain attribute is special is specified by the user in the description of the domain. The user will declare which planner will handle each attribute, the default being the general planner.

The itinerary planner is only asked to find a single solution. Indeed it would be too costly to provide IxTeT with several resolvers for a subproblem submitted to the itinerary planner. It would involve calculating all possible paths (with possible repeats of points visited) starting from one point and going through the places specified, respecting the given partial order. To enable IxTeT to backtrack on the choice of the path if needs be, a "continuation" resolver is to trigger a new call to the itinerary planner for another path (figure 4).

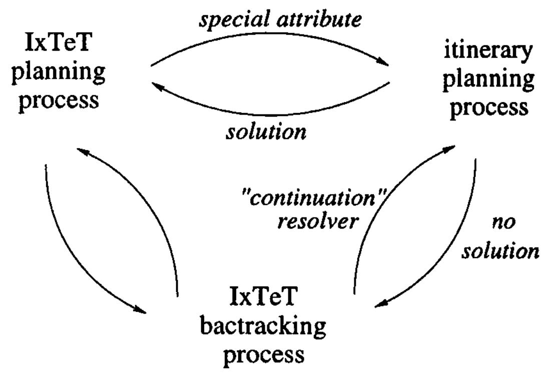


Figure 4: General process

The control

The itinerary planner needs to have complete information as it is designed to find a solution to IxTeT's entire movement problem. It is possible to know for certain, at a specific step of the planning process, that all possible flaws concerning the special attribute belong to the actual partial plan. No other flaws concerning that attribute will be introduced at a further step. This is due to the abstraction hierarchy.

The only way to introduce a flaw with attribute i, is to resolve a flaw with an attribute j of higher level. Only flaws of the current abstraction level may be resolved.
The abstraction level is updated by

- removing a node when all flaws of the attribute(s) of the node have been resolved
- adding a node which is a descendant of the node removed, and has no other ascendants in the last abstraction level

Let's suppose that the special attribute is at node 3 of the abstraction graph (figure 5). At a certain step of the planning process, the abstraction level is L: attributes of nodes 1 and 4 can both introduce flaws concerning the special attribute, as their abstraction classes are higher than that of the special attribute. A few steps further on in the planning process, L' could be the current abstraction level. Now no other flaw can contain an attribute at a higher level than the special attribute: all flaws of attributes of higher level have been resolved. So no other flaw resolution could introduce a new flaw concerning the special attribute (OMP property).

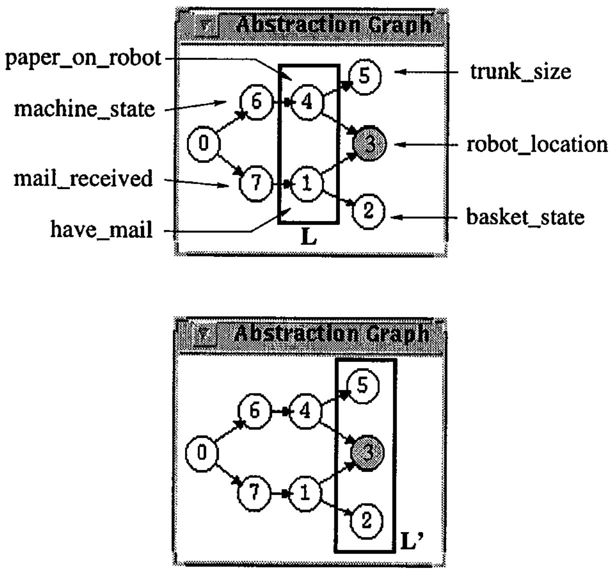


Figure 5: Abstraction evolution

When the node of the special attribute is introduced into the current abstraction level, we can thus be sure that no other flaw concerning this attribute will appear in IxTeT's partial plan. This is therefore the moment from which we may give over the control to the itinerary planner. This is done as soon as the flaw selected to be resolved contains the special attribute.

A limitation of this control method is that it cannot be used as such on a problem where there is another attribute in the same attribute class as the special attribute. This happens when attributes are interdependent. For example in the following two tasks, *robot_location* and *have_key* are in the same abstraction class (*robot_location* $\lhd$ *have_key* and *have_key* $\lhd$ *robot_location*).

```
task get_key(?key)(start,end)
{
    variable ?place;

    hold (robot_location(): ?place, (start,end));
    hold (key_position(?key): ?place, (start,end));
    event (key_position(?key): (?place,Robot), end);

    (end - start) in [00:01:00,00:03:00];
}

task go_to(?room1,?room2)(start,end)
{
    variable ?key;
    ROOM_KEYS(?key,?room2);

    hold (key_position(?key): Robot, (start,end));
    hold (robot_location(): ?room1, (start,end));
    event (robot_location(): (?room1,?room2), end);

    (end - start) in [00:04:00,00:09:00];
}
```

To overcome this limitation, the idea is to only partially resolve the flaws on the special attribute, enough to recursively introduce all the flaws on the attributes linked to it (*have_key* in the example). Let's call these attribute in the same abstraction class as the special one the *linked* attributes. By partial resolution, we mean that we will not attempt to add causal links, temporal constraints or variable constraints, nor call the path planner to complete the path until we have reached a certain point of the process. We will only add task resolvers, that will themselves add new flaws. We then resolve flaws on the linked attributes either by finding a trivial resolver (one already in the partial plan) or by using a new task resolver (which implies a new flaw on the special attribute). When only the flaws concerning the special attribute remain,we are back in the previous case: we can call the special planner to finish resolving these flaws. This may lead to new flaws on the problem attribute that may not have a trivial solution. We have to repeat the whole process until all flaws are solved.

Algorithm :

```
While there are flaws on either attribute
    1. while there are flaws on linked attributes
        1.1 resolution of linked flaws
        1.2 partial resolution of special flaws
    2. path planner
```

Experimental results

Several tests have been carried out using IxTeT combined with the itinerary planner. Here are some of the results obtained.
Examples 1, 2, 4 and 5 are based on a maintenance robot domain: an indoor mobile robot has to feed machines that run out of paper and to deliver arriving mail to the right offices.
Figure 6 gives a description of the domain for examples 4 and 5, figure 7 gives the solution plan found for example 4 (with temporal constraints between tasks, maximum and minimum starting and ending time-points). The topological graph associated to these examples is the one in figure 3.

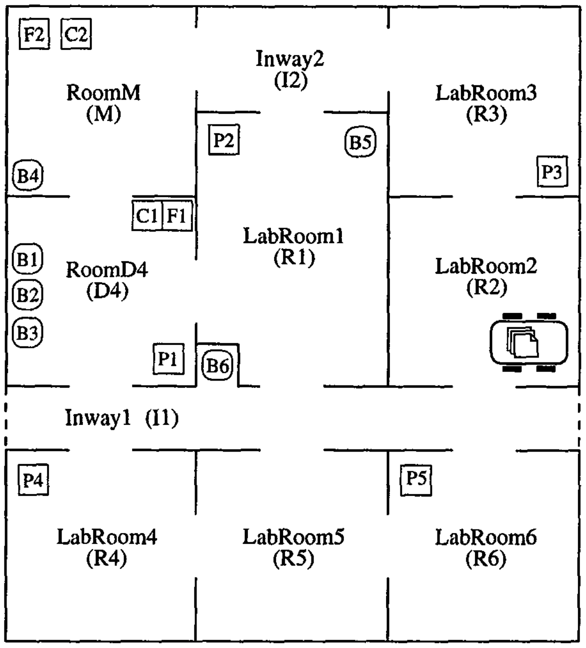
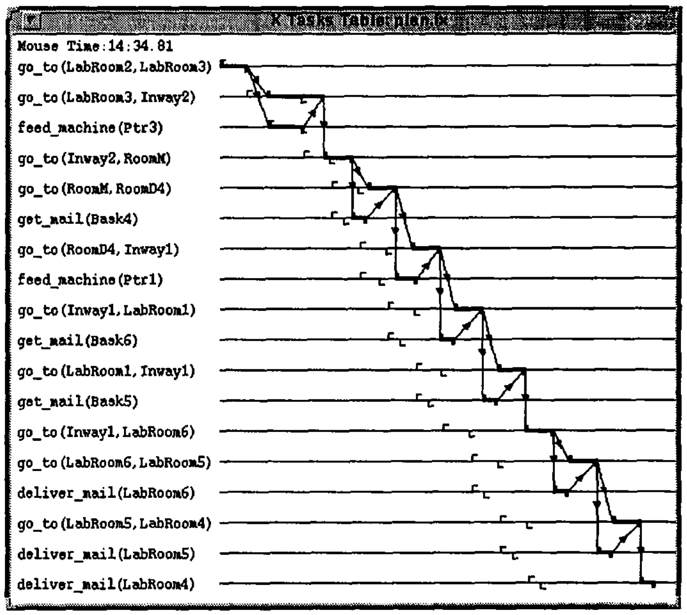


Figure 6: The initial state

Examples 3 and 6 again feature a mobile robot, this time having to move containers from one place to another.
Example 7 is based on a shopping expedition: a person



```
Mouse Time:14:34.81
go_to(LabRoom2,LabRoom3)
go_to(LabRoom3,Inway2)
feed_machine(Ptr3)
go_to(Inway2,RoomM)
go_to(RoomM,RoomD4)
get_mail(Bask4)
go_to(RoomD4,Inway1)
feed_machine(Ptr1)
go_to(Inway1,LabRoom1)
get_mail(Bask6)
go_to(LabRoom1,Inway1)
get_mail(Bask5)
go_to(Inway1,LabRoom6)
go_to(LabRoom6,LabRoom5)
deliver_mail(LabRoom6)
go_to(LabRoom5,LabRoom4)
deliver_mail(LabRoom5)
deliver_mail(LabRoom4)
```

Figure 7: The solution plan

has several things to buy in different shops, and an appointment to go to at a set time. An interesting point to note is that whereas IxTeT backtracks three times before finding a solution to this problem, IxTeT planning with the itinerary planner doesn't backtrack at all.

The following table gives the number of nodes and edges for each problem's topological graph, as well as the number of movement tasks over the total number of tasks in the solution plan.

Ex	Nodes	Edges	Tasks
1	4	8	3/5
2	5	10	4/6
3	27	148	4/6
4	10	28	11/20
5	10	28	9/17
6	27	148	8/12
7	11	24	15/20

The next table compares the time [2] spent in planning by IxTeT to the time spent when IxTeT cooperates with the itinerary planner. It also gives the gain achieved. The average gain over all examples is of 88%

[2]calculation times on a Sparc 10 machine

Ex	IxTeT	IxTeT +	Gain
1	0.5 s	0.2 s	60 %
2	2.4 s	0.4 s	83 %
3	14 s	1 s	93 %
4	71 s	6.5 s	91 %
5	94.9 s	4.1 s	96 %
6	124 s	3.5 s	97 %
7	178.8 s	4.3 s	98 %

We have compared the results we have obtained to those that Graphplan [1] obtains on the same examples. An exact comparison isn't possible, as IxTeT allows temporal planning, resource management, dynamic domains and gives partially ordered plans. We gave similar but reduced problems to Graphplan. Figure 8 shows a comparative graph of results obtained with IxTeT combined with an itinerary planner and Graphplan. The examples are ordered in growing execution-time for IxTeT. The graph shows that IxTeT with the itinerary planner compares quite well to Graphplan -a very efficient planner- on the examples tried out. Such a comparison shows that it is not necessary to give up expressiveness to reach a high level of efficiency.

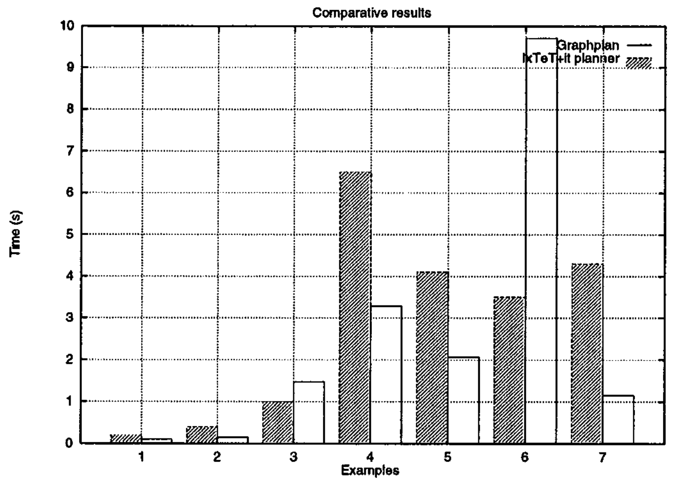


Figure 8: Comparative graph

Conclusion

We have proposed a method enabling to improve the performance of a general planner by using a domain-specific planner in order to perform part of the planning process. This idea has been tested on the general temporal planner IxTeT, integrating it with an itinerary planner. We have shown how to use IxTeT's abstraction hierarchy to control the integration process. The results obtained show a dramatic improvement of IxTeT's planning time, and compare favourably to the fast-planning Graphplan's results, even with its reduced expressiveness.

These results encourage us to think that the extension we intend to achieve in order to generalise this approach -by using several different domain-specific planners cooperating with IxTeT- should give us a good improvement of the general planner's overall performance. Another specific planner that would be interesting to integrate to IxTeT is a manipulation system for part assembly planning. We would then have a performing system, composed of an expressive general planner assisted by specialist planners for various robotics tasks: navigation or assembly. It would also enable us to make better use of the systems that we have at our disposition, by making them cooperate.

An important side effect of the contribution proposed here is that it permits to better and more easily interface the planner with the supervision and execution control system of the robot. But this issue, currently carried out in our research group, is outside of the scope of this paper.

References

[1] M. Furst A. Blum. Fast Planning Through Graph Analysis. *Artificial Intelligence*, 90:281–300, 1997.

[2] R.Aylett G.Petley P.Chung J.Soutter A.Rushton. Planning and Chemical Plant Operating Procedure Synthesis: a Case Study. In *Fourth European Conference on Planning*, 1997.

[3] F. Kabanza F. Bacchus. Using Temporal Logic to Control Search in a Forward Chaining Planner. In *New Directions in IA planning, IOS Press*, pages 141–153, 1995.

[4] P. Laborie F. Garcia. Hierarchisation of the Search Space in Temporal Planning. In *New Directions in IA planning, IOS Press*, pages 217–231, 1995.

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

[6] M. Ghallab and A. Mounir-Alaoui. The Indexed Time Table Approach for Planning and Acting. In *Proc. NASA Conference on Space Telerobotics*, February 1989.

[7] M. Ghallab and T. Vidal. Focusing on a Subgraph for Managing Efficiently Numerical Temporal Constraints. In *Proceedings FLAIRS-95*, 1995.