TALplanner: A Temporal Logic Based Planner

Patrick Doherty and Jonas Kvarnström

Abstract

Talplanner is a forward-chaining planner which utilizes domain-dependent knowledge to control search in the state space generated by action invocation. The domain-dependent control knowledge, background knowledge, plans and goals are all represented using formulas in a temporal logic called TAL. TAL has been developed independently as a formalism for specifying agent narratives and reasoning about them. In the recent AIPS'00 planning competition, TALplanner exhibited impressive performance, winning the outstanding performance award in the domain dependent planning competition. In this article, we provide an overview of TALplanner.

This article is an invited submission to AI magazine, 2000.

Introduction

TALplanner (Doherty & Kvarnström 1999; Kvarnström, Doherty, & Haslum 2000; Kvarnström & Doherty 2000a; Kvarnström & Doherty 2000b) participated in the recent planning competition at the 5th International Artificial Intelligence Planning and Scheduling Conference, which took place in Breckinridge, Colorado, May 2000. TALplanner received the Outstanding Performance award in the domain dependent planning competition and 1st place in the Miconic 10 elevator control domain competition sponsored by Schindler Lifts Ltd. For the domains used in the competition, TALplanner exhibited remarkable performance in comparison to many of the other state of the art planners which participated in the competition.

Talplanner is a forward-chaining planner which utilizes domain-dependent knowledge to control search in the state space generated by action invocation. The domain-dependent control knowledge, background knowledge, plans and goals are all represented using formulas in a temporal logic called TAL. TAL has been developed independently as a formalism for specifying agent narratives and reasoning about them. A narrative consists of a specification of fluents which hold at various points in time, causal dependencies which relate fluent change, action types which characterize action occurrences which may be invoked by an agent and domain constraints which characterize background knowledge. A logical model for a narrative describes a linear sequence of states where fluents have unique values in each state. A plan is viewed as a narrative, plan operators are viewed as action types and domain-dependent control knowledge and goals as temporal formulas entailed by the generated narrative.

Although forward-chaining planners generally suffer from a lack of goal-directedness when compared to other types of planners such as regression-based or partial-order planners, for many domains the use of explicitly represented domain-dependent knowledge more than compensates for this deficiency. More significantly, a forward chaining planner always has a complete description of the past and current states, which facilitates the use of complex operator types with complex preconditions and conditional effects.

The use of a 1st-order temporal logic language is well suited to compactly represent both the complex operator features and the control knowledge used to prune the search space. This representation is highly amenable to the syntactic transformations used in various types of optimizations associated with the planning algorithm. In addition, the use of logic for representation provides a natural semantics for plans, goals and control knowledge.

How did TALplanner come about? As stated previously, we have spent a number of years developing logical representations of agent behaviors in the form of narratives using temporal logic. More recently, we have been involved in an unmanned aerial vehicle project which includes development of deliberative/reactive systems to support autonomous behavior. One of the central components of the architecture is a planning module which more often than not must generate plans in a timely or anytime manner. In addition, the planner must have the capability to reason about explicit time and represent actions with complex interactions and duration.

After surveying the planning literature, we found few if any planning approaches which had the potential for dealing with the many constraints associated with the UAV project. There was one exception though, TLplan (Bacchus & Kabanza 1996; 1998; 2000). We were immediately struck by the simplicity and elegance of the approach in addition to its performance. TLplan uses a modal tense logic to represent domain-dependent control knowledge and their forward chaining algorithm is based on the use of formula progression, a technique similar to that used in tableau theorem provers for tense logics.

In order to test the feasibility of the approach, we implemented an initial version of TALplanner which translated between temporal formulas in our formalism and tense logic formulas and used formula progression. The results were promising. In fact, this implementation was faster then TLplan when tested using a number of benchmarks from the AIPS'98 competition. On the other hand, we were less satisfied with the need to translate and use a formula progression algorithm. Consequently, we began experimentation with a different approach which evaluates TAL formulas directly without the use of formula progression. This latest version of TALplanner is substantially different from TLplan and the performance is markedly better than the original approach both in terms of time and space complexity. In addition, the relation between TALplanner and TAL is much more comprehensive. This offers methodological advantages as TALplanner is incrementally extended in the future.

In the rest of this article, we provide an overview of TALplanner. We begin by introducing the temporal logic TAL and then proceed to the methodological framework used to develop TALplanner. We then describe a robotics gripper domain example followed by a presentation of the planning algorithm and its operation. This is followed by a discussion about optimization techniques used in TALplanner. A comparison between TALplanner and TLplan is made using a number of benchmark examples from the AIPS'98 planning competition which are similar to those used in AIPS'00 and equally challenging. We conclude with a discussion about additional extensions to TALplanner not described in this article and some future directions for research.

TAL: Temporal Action Logics

TAL, Temporal Action Logics (Doherty *et al.* 1998), is a family of narrative-based first-order temporal logics developed for reasoning about action and change in incompletely specified dynamic worlds. The TAL logics share a 1st-order base language $\mathcal{L}(\text{FL})$, used for formally reasoning about narratives. From a knowledge engineering perspective, viewing narratives as sets of 1st-order formulas lacks structure and modularity. Instead, we developed a higher-level macro language called $\mathcal{L}(\text{ND})$ which permits structured representation of narratives using labeled statements. One can automatically transform a set of statements in $\mathcal{L}(\text{ND})$ into a set of 1st-order formulas in $\mathcal{L}(\text{FL})$.

Because our worlds are incompletely specified such issues as the frame, qualification and ramification problems arise. We provide solutions to these problems using a combination of representational techniques and Circumscription (McCarthy 1980) which enable the encoding of flexible closed world and persistence assumptions. Efficient inference from the resulting theories is made possible via the use of quantifier elimination techniques, a subject for another paper. Currently, TALplanner only requires the evaluation of a TAL formula in a model where the domain is restricted to be finite. In (Doherty & Kvarnström 1999; Kvarnström & Doherty 2000a) we presented a version of $\mathcal{L}(\text{ND})$ called $\mathcal{L}(\text{ND})^*$, suitable for modeling planning domains and problem instances. Compared to other $\mathcal{L}(\text{ND})$ versions, $\mathcal{L}(\text{ND})^*$ contains two additional statement classes for goals and control rules. In a following section, we will provide examples of some of these statement classes.

Framework and Methodology

Figure 1 provides an abstract overview of TALplanner's architecture in terms of input and output and the relation between TAL and TALplanner. The figure also provides a means of understanding the research methodology used in the development of TALplanner.

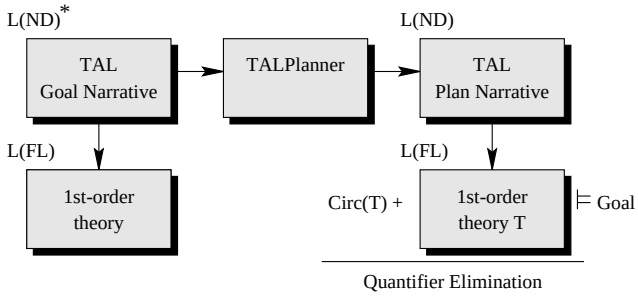

Figure 1: TAL/TALplanner relation

In figure 1, TALplanner takes a TAL goal narrative in $\mathcal{L}(\text{ND})^*$ as input. The planner translates the goal narrative into a suitable internal representation and then searches for a plan, an operator sequence satisfying the goal statement and the control rules. If a plan exists, the result is a new narrative in $\mathcal{L}(\text{ND})$ where goals and control rules have been removed and a set of TAL action occurrences (corresponding to plan steps) has been added. Observe that one can use standard inference techniques or techniques specific to TAL to reason about both the input goal narrative and the output narrative. The output narrative is always guaranteed to entail the original goal and the domain dependent control knowledge.

The methodology we use to incrementally extend TALplanner is based on our experience with TAL. TAL serves as a reference formalism for TALplanner, where the language used to represent narratives in TAL may be viewed as a rich and expressive plan representation language. The plan synthesis algorithm associated with TALplanner is incrementally extended by increasing the expressivity of the plan operators and other narrative statement classes. The semantics and understanding of the extensions is always grounded in the formal semantics associated with TAL.

Observe that TALplanner does not subscribe to the *planning as theorem proving* paradigm. Instead, the associated logic TAL, serves as a formal plan specification language with an associated semantics for understanding the intricacies of operator invocation in incompletely specified world domains. The implementation reflects this by constructing partial models which are incremented as plan operators are added and filtered as control formulas are evaluated in these partial models.

A Gripper Domain Example

We will use a simple gripper domain as an example. In this domain, a robot, Robby, can move objects between a number of rooms. For simplicity, we will assume Robby only has a single gripper. Each object is initially in a room, and the goal requires some objects to end up in certain rooms. We model this using the sorts boolean = {true,false}, obj containing objects and room containing locations and the propositional (boolean) state variables at-robby(room), at(obj, room), carry(obj) and free. Moving takes three timepoints, while picking up and dropping objects takes a single timepoint.

The following three operator descriptions are shown with the internal representation used in TAlplanner:

```
#operator move(from, to) :at t
   :precond [t] at-robby(from)
   :effects [+3] at-robby(to) := true,
            [+3] at-robby(from) := false
#operator pick(ball, room) :at t
   :precond [t] at(ball, room) ∧ at-robby(room) ∧ free
   :effects [+1] carry(ball) := true,
            [+1] at(ball, room) := false,
            [+1] free := false
#operator drop(ball, room) :at t
   :precond [t] carry(ball) ∧ at-robby(room)
   :effects [+1] carry(ball) := false,
            [+1] at(ball, room) := true,
            [+1] free := true
```

In the narrative language $\mathcal{L}$(ND), the move operator belongs to the narrative statement class *action type* and would be represented as follows:

acs $[t, t']\ \mathsf{move}(from, to) \rightsquigarrow$
$t' = t + 3 \wedge$
$([t]\ \text{at-robby}(from) \rightarrow$
$R([t + 3]\ \text{at-robby}(to) \hat{=} \text{true}) \wedge$
$R([t + 3]\ \text{at-robby}(from) \hat{=} \text{false}))$

In the logical language L(FL), the move operator would be represented as follows:

$\forall t, t', from, to.Occurs(t, t', \mathsf{move}(from, to)) \rightarrow$
$t' = t + 3 \wedge$
$(Holds(t, \text{at-robby}(from), \text{true}) \rightarrow$
$Holds(t + 3, \text{at-robby}(to), \text{true}) \wedge$
$Occlude(t + 3, \text{at-robby}(to)) \wedge$
$Holds(t + 3, \text{at-robby}(from), \text{false}) \wedge$
$Occlude(t + 3, \text{at-robby}(from)))$

Similar translations are used for the other narrative statement classes.

For the following problem instance, we can assume the following sorts: room $=$ {roomA, roomB}, obj $=$ {ball1, ball2}.

A plan problem instance description contains a specification of initial state, eg. $[0]\ \forall obj[\text{at}(obj, \text{roomA}) \wedge \neg\text{carry}(obj)] \wedge \text{at-robby}(\text{roomA})$, and a goal state, eg. $\forall obj[\text{at}(obj, \text{roomB})]$.

Control Formulas for the Gripper Domain

For the gripper domain, we will use the following three control statements specified in $\mathcal{L}$(ND).

First, suppose Robby is carrying an object that should be in the current room. Clearly, he should not move before putting it down – if he does, he will have to return later. In other words, he should remain in the same location in the following state:

$\forall t, r, o.\ [t]\ \text{at-robby}(r) \wedge \text{carry}(o) \wedge \text{goal}(\text{at}(o, r)) \rightarrow$
$[t + 1]\ \text{at-robby}(r)$

Second, a similar condition applies if Robby's gripper is free and there is an object here that should be moved to another room. Again, in order to avoid having to return later, he should not move until he has picked it up:

$\forall t, r, o.\ [t]\ \text{free} \wedge \text{at-robby}(r) \wedge \text{at}(o, r) \wedge$
$[t]\ \exists r'[r' \neq r \wedge \text{goal}(\text{at}(o, r'))] \rightarrow$
$[t + 1]\ \text{at-robby}(r).$

Third, only pick up an object if the goal requires it to be in another room:

$\forall t, o.\ [t]\ \neg\text{carry}(o) \wedge$
$[t]\ \forall r, r'[\text{at}(o, r) \wedge \text{goal}(\text{at}(o, r')) \rightarrow r = r'] \rightarrow$
$[t + 1]\ \neg\text{carry}(o).$

The task of specifying control statements is currently the responsibility of the domain designer. For many domains, the process is intuitive and straightforward. We imagine that for other domains, the process will be quite complex and finding a means of automatically generating at least some of the control statements is highly desirable and a challenging research issue.

TALplanner

In the previous section, we provided a description of the components in a TAL goal narrative using the Gripper Domain example. TALplanner takes a TAL goal narrative as input, and generates a new narrative where a set of timed action occurrences has been added. Internally, however, TALplanner is basically a forward chaining planner, searching through the space of states reachable from the initial state. In order to be able to describe this search space in more detail, we must first provide a few definitions.

Due to the use of actions with non-unit duration, plans cannot be simple sequences of operators but must also contain timing information. This information is provided as *timed operator instances* of the form $[s, t]\ o$, denoting the invocation of the operator instance o between times s and t, where $s < t$.

An *executable operator sequence* is a tuple of timed operator instances with the following constraints. First, the empty tuple is an executable operator sequence. Second, given a sequence of n operators ending in $[t_{n-1}, t_n]\ o_n$, its successors are exactly those sequences adding one new timed operator instance $[t_n, t_{n+1}]\, o_{n+1}$ such that o_{n+1} is applicable at $[t_n, t_{n+1}]$ (where $t_n = 0$ if $n = 0$).

A *plan* is an executable operator sequence that corresponds to an infinite state sequence that satisfies all control rules as well as the goal.

These definitions induce a search tree where the root is labeled with the empty sequence and the children of a node labeled l are labeled with the successors of l. Clearly, this search tree must contain all plans. Therefore, a complete planner can be generated by using a complete search algorithm such as breadth first search or iterative deepening.

Conceptually, each node in the search space contains an

executable operator sequence. Each node represents a partial model in TAL for the narrative being constructed. Each partial model contains a sequence of states (or prefix to the final state sequence) which will entail the goal statement if successful. Nodes are filtered if they do not satisfy domain dependent control formulas and other constraints associated with the narrative. Figure 2 provides a diagram of the search and pruning process. The initial node is a goal narrative, the internal nodes in the tree are representations of partial TAL models (executable operator sequences). The goal node is transformed into a standard TAL narrative containing a generated sequence of plan operator instances.

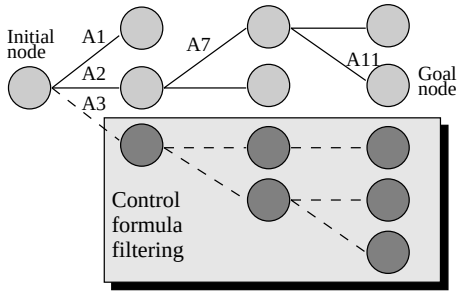

Figure 2: Pruning

Below, we provide an abstract description of the TALplanner algorithm which generates the search space depicted in figure 2 above.

Input: An initial goal narrative $\mathcal{GN}$ in $\mathcal{L}(\text{ND})^*$.
Output: A plan narrative $\mathcal{N}_p$ in $\mathcal{L}(\text{ND})$ which entails the goal and the control rules.

```
1  procedure TALplan(GN)
2  Open ← ⟨⟨0, GN⟩⟩   // Stack (depth-first search)
3  while Open ≠ ⟨⟩ do
4    ⟨τ, GN⟩ ← pop(Open)
5    N ← (GN minus goals and control rules)
6    if the pruning constraints are satisfied in N then
7      if the state goal and control rules are satisfied return N
8      if cycle checking disabled or there is no cycle then
9        for every action [τ, τ'] o applicable at τ do
10         push ⟨τ', GN ∪ {[τ, τ'] o}⟩ on Open
11 fail
```

The algorithm itself is straightforward; one forward chains on operator invocations using (in this case) a depth-first search strategy. Branches in the search space are pruned using *pruning constraints* which are derived from the set of control statements supplied to the planning algorithm. The bottleneck in the algorithm is checking when a formula is satisfied in a narrative model. We consider these issues in the next section.

The expressiveness of the operators and use of other narrative statement classes was highly constrained in the competition. Plan operators were restricted to be deterministic, complete information about initial state was assumed and actions were single step. In spite of this, the version of TALplanner used in the competition allows for order-sorted finite value domains for fluents (both boolean and non-boolean domains are allowed), fluents (state variables) can take arbitrary numbers of arguments, action types (operators) can be context dependent with arbitrary preconditions and conditional effects. They can also have arbitrary integer duration, and effects are not limited to the final timepoint but can take place anywhere in the duration of the action. Arbitrary goals are allowed, including existential goals. TAL provides a semantics for all of these extensions.

Optimization Techniques

Where is the magic in the algorithm which would explain its extraordinary performance in the domains used in the competition? Clearly, it is not the algorithm itself which is a forward chaining algorithm with a depth first search strategy.

The ability to represent complex domain dependent control knowledge compactly in a 1st-order language without compiling to a propositional representation is one of the answers.

Since the bottleneck in the algorithm is to evaluate a 1st-order formula in a data structure representing a TAL model, syntactic transformations which simplify the original control formulas into equivalent, but more efficiently evaluated formulas is another answer. We call such formulas *pruning constraints*.

Pruning constraints themselves can be optimized by taking advantage of information implicit in plan operators. Each pruning constraint is analyzed separately for each operator type in a domain, under the assumption that some instance of that operator has just been invoked. Although this is done during a pre-processing phase, where exact arguments and timepoints are not yet known, the operator descriptions contain considerable information about the states in which the pruning constraints will eventually be evaluated. In some cases, the process completely removes pruning constraints for certain operators, thus saving the time it takes to evaluate the constraint.

In many cases, the operator-specific analysis described above results in pruning constraints where some conjuncts, or even the entire constraints, only refer to the invocation state of the operator. TALplanner moves such conjuncts into the precondition of the operator, automatically generating so called *precondition control* (Bacchus & Ady 1999). This improves performance significantly, since control rule violations can be detected before the planner even attempts to invoke an operator.

TALplanner allows the domain designer to specify a set of *assertions*, conditions that must necessarily hold in any state sequence reachable from any valid initial state. In the gripper domain, one such assertion would be $\forall t, room, room'.[t]\ \text{at-robby}(room) \wedge \text{at-robby}(room') \rightarrow room = room'$: Robby can never be in two rooms at once.

Assertions are mainly used during the analysis phase. For example, when analyzing pruning constraints relative to operators, assertions can be used to infer additional information about an invocation state given what is explicitly stated in a precondition. This sometimes allows TALplanner to simplify pruning constraints further, improving the chances of converting constraints into precondition control.

In figure 3 below, the preprocessing phase used in TALplanner is depicted. The optimizer takes the control rules, assertions and plan operators as input and outputs a set of optimized pruning constraints per operator. The process is fully automated and will be described in detail in a forthcoming paper. Note that the complete pre-processing phase normally takes about 10-100 milliseconds. Future versions of TALplanner may integrate domain analysis techniques to automatically generate assertions.

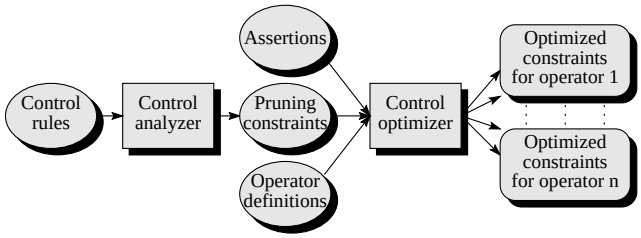

Figure 3: Optimization

Comparison with TLplan

TLplan did not compete in this year's competition to avoid any conflict of interest due to Fahiem Bacchus role as organizer of the competition. Although he did a fantastic job organizing and running the competition, TLplan's absence was unfortunate since it is one of the fastest planner's around and surely would have done well in the competition. In addition, TALplanner uses the same planning paradigm and comparisons between the two are of some interest. In the following, we make an attempt at comparing the two planners via the use of benchmark problems used in the AIPS'98 competition.

As stated in the beginning of the article, the first prototype implementation of TALplanner basically tried to emulate TLplan by using translations of TAL formulas into tense logical formulas and using formula progression techniques. We call this version of TALplanner *TALplanner/progression* in the diagrams below. The second prototype implementation of TALplanner based on direct evaluation of TAL formulas and a preprocessing optimization phase was used in the competition. We call this version of TALplanner *TALplanner/evaluation* in the diagrams below.

All benchmarks were run on the same 333 MHz Pentium II computer running Windows NT 4.0 SP3, using 256 MB of memory. The machine is quite slow by current standards. For comparison with TLPLAN, we used the precompiled version that can be downloaded from http://www.lpaig.uwaterloo.ca/~fbacchus/. TALplanner is written in Java, and we used TALplanner 2.741 with the Java Development Kit 1.2.2-001 and the HotSpot virtual machine (1.0fcs), both of which can be downloaded from http://java.sun.com. In all cases, we made sure that the computer was very lightly loaded and that it was never swapping.

Benchmark Results

We use two of the domains chosen for both the AIPS'98 and AIPS'2000 competitions, the logistics and blocks domains. All 30 problems from the logistics domain were taken from the AIPS'98 competition. All 43 problems from the blocks domain were randomly generated by us. For TLplan, the domain descriptions and control rules used are those from the original description used by Bacchus and Kabanza (available in the TLplan software distribution). For TALplanner/progression, the domain descriptions use exactly the same modal control formulas. For TALplanner/evaluation, the domain descriptions use control formulas based on the modal control formulas used by Bacchus and Kabanza. The rules are modified somewhat to allow TALplanner's optimizer to detect additional optimization opportunities.

Logistics

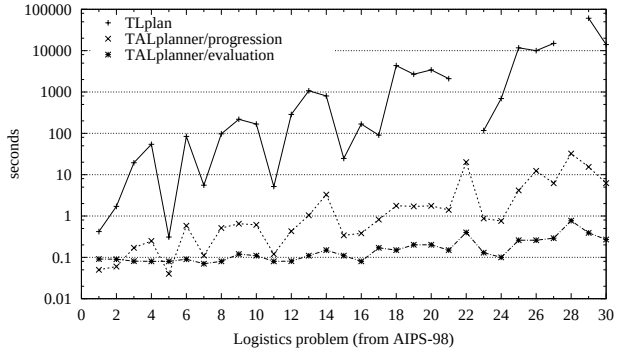

Figure 4: Logistics Time

Figure 4 shows how much time the planners needed for the logistics problems. TLplan creates the first few plans in around a second, but needs over 15 hours for some problems and cannot solve two of the problems with 256 MB of memory. TALplanner is considerably faster even using progression. With evaluation, TALplanner solves the largest problem (number 28) in under 0.8 seconds and never exceeds the 4 MB of heap space automatically allocated by the Java virtual machine. TLplan could not solve this problem at all (with the amount of memory available). The largest problem it could solve was number 29, which required > 60000 seconds, compared to 15.422 seconds for modal TALplanner and 0.391 seconds for TALplanner using evaluation.

Figure 5 shows how many worlds (states) the planners expanded for the logistics problems.

Blocks

Figure 6 shows how much time the planners needed for the blocks problems. TLplan creates the first few plans in under 10 seconds, but needs almost nine hours for some problems and cannot solve the larger problems with 256 MB of memory. TALplanner/progression is considerably faster and handles considerably larger problems. With evaluation, TALplanner solves the largest problem (number 43, with 5000 blocks and $>$ 15000 operators) in under 20 seconds

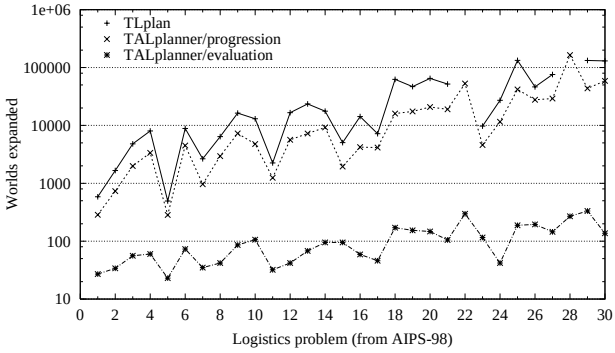

Figure 5: Logistics Worlds

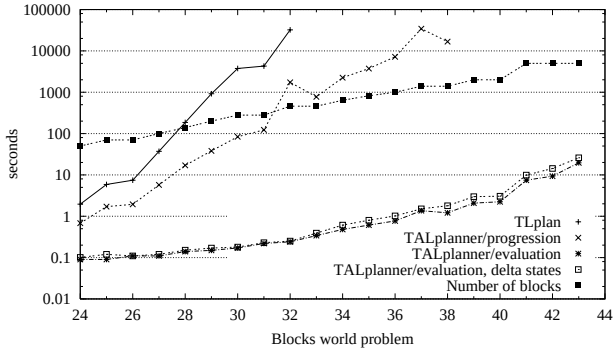

Figure 6: Blocks Time Delta

using 63MB of memory. Clearly, 5000 blocks and 15000 operators is nowhere near the limit. TALplanner/evaluation with delta states[1] is somewhat slower, requiring up to 26 seconds, but only uses 27MB of memory. The first 34 problems are solved using less than the 4MB heap space allocated by the Java VM.

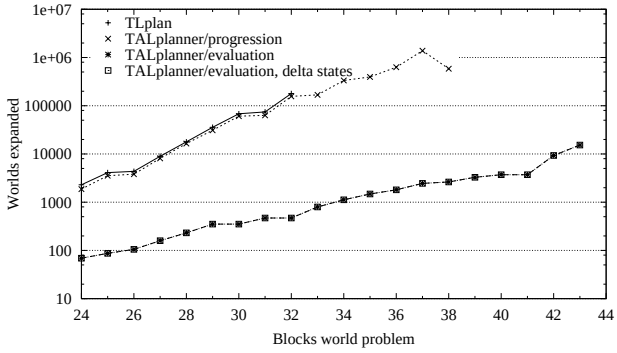

Figure 7: Blocks Worlds Delta

Figures 7 shows how many worlds (states) the planners expanded for the blocks problems.

Additional Extensions and Future Work

Clearly, TALplanner has shown some potentially promising results with an ability to scale up for larger problems. It remains to be seen how well the planner works for other domains, especially where the constraints on complete states and deterministic actions are lifted. TALplanner has been extended for concurrent actions and resources (Kvarnström, Doherty, & Haslum 2000). See (Kvarnström & Doherty 2000a) for a more detailed description of the sequential and concurrent versions of the planner. Current work with TALplanner includes extending it to work with incomplete information states and applying it to the UAV domain.

Acknowledgments

This research is supported in part by the Swedish Research Council for Engineering Sciences (TFR) and the Wallenberg Foundation, Sweden.

References

Bacchus, F., and Ady, M. 1999. Precondition control. Available at ftp://newlogos.uwaterloo.ca/pub/bacchus/BApre.ps.gz.

Bacchus, F., and Kabanza, F. 1996. Using temporal logic to control search in a forward chaining planner. In Ghallab, M., and Milani, A., eds., *New Directions in AI Planning*. ISO Press. 141–153.

Bacchus, F., and Kabanza, F. 1998. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence* 22:5–27.

[1]Delta states is a more memory efficient, but less time efficient state space representation.

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116:123–191.

Doherty, P., and Kvarnström, J. 1999. TALplanner: An empirical investigation of a temporal logic-based forward chaining planner. In *Proc. of the 6th Int'l Workshop on Temporal Representation and Reasoning*, 47–54.

Doherty, P.; Gustafsson, J.; Karlsson, L.; and Kvarnström, J. 1998. TAL: Temporal Action Logics – language specification and tutorial. *Linköping Electronic Articles in Computer and Information Science* 3(15). Available at http://www.ep.liu.se/ea/cis/1998/015.

Kvarnström, J., and Doherty, P. 2000a. Talplanner: A temporal logic based forward chaining planner. *Annals of Mathematics and Artificial Intelligence*. Accepted for publication.

Kvarnström, J.; Doherty, P.; and Haslum, P. 2000. Extending TALplanner with concurrency and resources. In *Proceedings of the 14th European Conference on Artificial Intelligence*.

Kvarnström, J., and Doherty, P. 2000b. TALplanner home page. http://www.ida.liu.se/labs/kplab/talplanner.

McCarthy, J. 1980. Circumscription – a form of non-monotonic reasoning. *Artificial Intelligence* 13:27–39.