

TALPLANNER

A Temporal Logic-Based Planner

Patrick Doherty and Jonas Kvarnström

■ TALPLANNER is a forward-chaining planner that 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, which has been developed independently as a formalism for specifying agent narratives and reasoning about them. In the Fifth International Artificial Intelligence Planning and Scheduling Conference 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.

TALPLANNER¹ (Doherty and Kvarnström 1999; Kvarnström and Doherty 2001; Kvarnström, Doherty, and Haslum 2000) participated in the recent planning competition at the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS'00), which took place in Breckenridge, Colorado, in April 2000. TALPLANNER received the Outstanding Performance Award in the Domain-Dependent Planning Competition and first 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 that participated in the competition.

TALPLANNER is a forward-chaining planner that 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, which has been developed independently as a formalism for specifying agent narratives and reasoning about them. A narrative consists of a specification of fluents that hold at various points in time, causal dependencies that relate fluent change, action types that characterize action occurrences that can be invoked by an agent, and domain constraints that 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 first-order temporal logic language is well suited for compactly representing 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 optimization 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 (UAV) project that includes development of deliberative-reactive systems to support autonomous behavior. One of the central components of the architecture is a planning module that 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 that had the potential for dealing with the many constraints associated with the UAV project. There was one exception, though: TLPLAN (Bacchus and Kabanza 2000, 1998, 1996). 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.

To test the feasibility of the approach, we implemented an initial version of TALPLANNER that 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 than TLPLAN when tested using a number of benchmarks from the AIPS'98 competition. However, we were less satisfied with the need to translate and use a formula progression algorithm. Consequently, we began experimentation with a different approach that 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, which 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, which 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 first-order base language $L(FL)$, used for formally reasoning about narratives. From a knowledge engineering perspective, viewing narratives as sets of first-order formulas lacks structure and modularity. Instead, we developed a higher-level macro language called $L(ND)$ that permits structured representation of narratives using labeled statements. One can automatically transform a set of statements in $L(ND)$ into a set of first-order formulas in $L(FL)$.

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) that enable the encoding of flexible closed-world and persistence assumptions. Efficient inference from the resulting theories is made possible with the use of quantifier elimination techniques, a subject for another article. Currently, TALPLANNER only requires the evaluation of a TAL formula in a model where the domain is restricted to be finite. In Doherty and Kvarnström (1999) and Kvarnström and Doherty (2001), we presented a version of $L(ND)$ called $L(ND)^*$, suitable for modeling planning domains and problem instances. Compared to other $L(ND)$ versions, it contains two additional statement classes for goals and control rules. In a following section, we 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.

In figure 1, TALPLANNER takes a TAL goal nar-

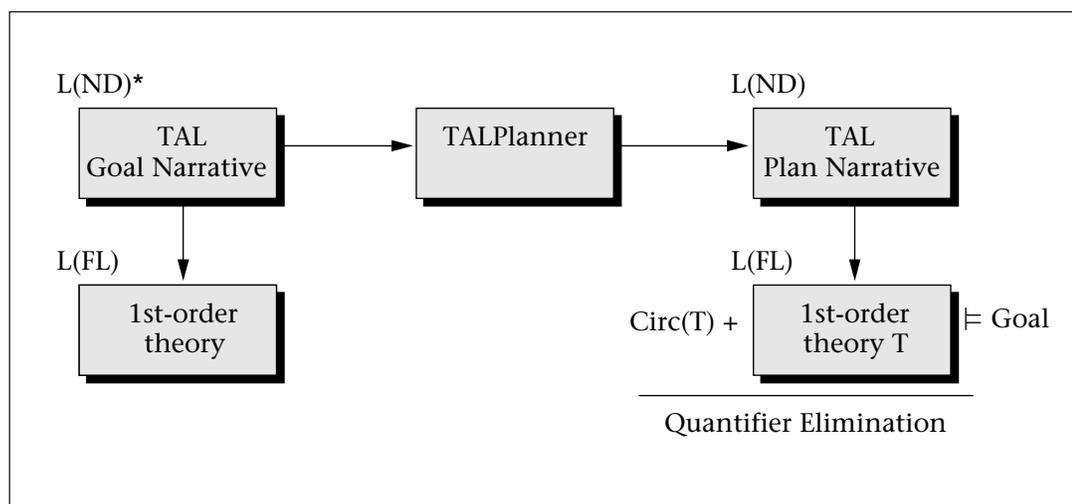


Figure 1. TAL/TALPLANNER Relation.

rative in $L(ND)^*$ 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 $L(ND)$ 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 can 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 are 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 view by constructing partial models that are incremented as plan operators are added and filtered as control formulas are evaluated in these partial models.

A Gripper Domain Example

We 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 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 $\text{boolean} = \{\text{true}, \text{false}\}$, obj containing objects and room containing locations, and the propositional (Boolean) state variables $\text{at-robby}(\text{room})$, $\text{at}(\text{obj}, \text{room})$, $\text{carry}(\text{obj})$, and free . Moving takes three time points, and picking up and dropping objects takes a single time point.

In figure 2, three operator descriptions are shown with the $L(ND)^*$ syntax used in TALPLANNER.

In the narrative language $L(ND)$, the move operator belongs to the narrative statement class action type and would be represented as shown in figure 3.

In the logical language $L(FL)$, the move operator would be represented as shown in figure 4.

Similar translations are used for the other narrative statement classes.

The definition of a specific problem instance contains a specification of the sorts, for example, $\text{room} = \{\text{roomA}, \text{roomB}\}$ and $\text{obj} = \{\text{ball1}, \text{ball2}\}$. It also specifies an initial state, for example, $[0] \forall \text{obj} [\text{at}(\text{obj}, \text{roomA}) \wedge \neg \text{carry}(\text{obj})] \wedge \text{at-robby}(\text{roomA})$, and partially or completely describes a goal state, for example, $\forall \text{obj} [\text{at}(\text{obj}, \text{roomB})]$.

Control formulas for the gripper domain: For the gripper domain, we use the following three control statements specified in $L(ND)^*$. These formulas must hold in any plan generated by 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

```

Figure 2. Three Operator Descriptions with the $L(ND)^*$ Syntax Used in TALPLANNER.

```

acs [t, t'] move(from, to) ~>
t' = t + 3 ∧
([t] at-robby(from) →
R([t + 3] at-robby(to) Δ true) ∧
R([t + 3] at-robby(from) Δ false))

```

Figure 3. Move Operator in the Narrative Language $L(ND)$.

```

∀ t, t', from, to. Occurs(t, t', move(from, to)) →
t' = t + 3 ∧
(Holds(t, at-robby(from), true) →
Holds(t + 3, at-robby(to), true) ∧
Occlude(t + 3, at-robby(to)) ∧
Holds(t + 3, at-robby(from), false) ∧
Occlude(t + 3, at-robby(from)))

```

Figure 4. Move Operator in the Logical Language $L(FL)$.

First, suppose ROBBY is carrying an object that should be in the current room. Clearly, it should not move before putting it down; if it does, it has to return later. In other words, it 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, to avoid having to return later, it should not move until it 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. To be able to describe this search space in more detail, we must first provide a few definitions.

Because of the use of actions with nonunit 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) that will entail the goal statement if successful.

TALPLANNER analyzes the control formulas, which must hold in the final plan, and extracts pruning constraints that must hold in each ancestor of a plan. Any node that violates a pruning constraint is immediately filtered out.

Figure 5 provides a diagram of the search and pruning process. The initial node is a goal narrative, and 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.

In figure 6, we provide an abstract description of the TALPLANNER algorithm, which generates the search space depicted in figure 5.

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 that are automatically derived from the control statements. One of the bottlenecks 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 the use of other narrative statement classes were 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 these restrictions, 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, and 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 time point but can take place anywhere in the duration of the action. Arbitrary goals are allowed, including existential goals. TAL provides a semantics for all these extensions.

Optimization Techniques

Where is the magic in the algorithm that would explain its extraordinary performance in the domains used in the competition? Clear-

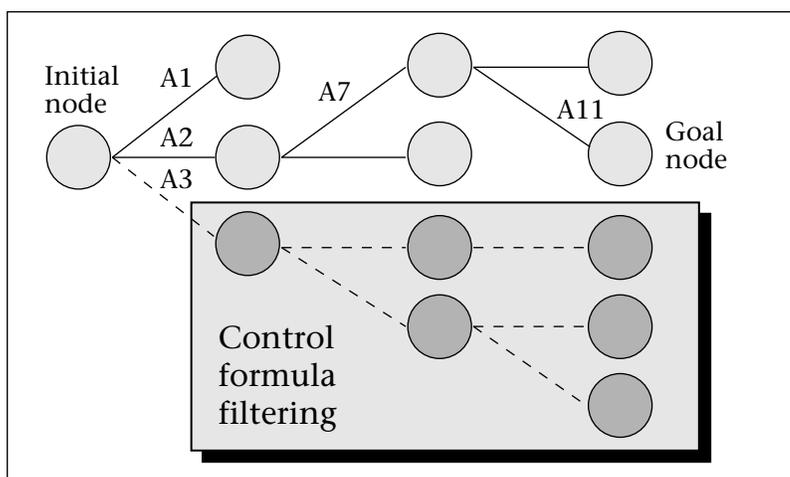


Figure 5. Pruning.

Input: An initial goal narrative \mathcal{G}_N in $L(\text{ND})^*$.

Output: A plan narrative $\mathcal{N}p$ in $L(\text{ND})$ that entails the goal and the control rules.

```

1 procedure TALPLAN( $\mathcal{G}_N$ )
2   Open  $\leftarrow \langle\langle 0, \mathcal{G}_N \rangle\rangle$  // Stack (depth-first search)
3   while Open  $\neq \langle\rangle$  do
4      $\langle\tau, \mathcal{G}_N\rangle \leftarrow \text{pop}(\text{Open})$ 
5      $\mathcal{N} \leftarrow (\mathcal{G}_N \text{ minus goals and control rules})$ 
6     if the pruning constraints are satisfied in  $\mathcal{N}$  then
7       if the state goal and control rules are satisfied return  $\mathcal{N}$ 
8       if cycle checking disabled or there is no cycle then
9         for every action  $[\tau, \tau]_o$  applicable at  $\tau$  do
10          push  $\langle\tau', \mathcal{G}_N \cup \{[\tau, \tau]_o\}\rangle$  on Open
11 fail

```

Figure 6. Abstract Description of the TALPLANNER Algorithm.

ly, it is not the algorithm itself, which, as described previously, is a forward-chaining algorithm with a depth-first search strategy.

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

Because one of the bottlenecks in the algorithm is to evaluate a first-order formula in a data structure representing a TAL model, syntactic transformations that simplify the automatically extracted pruning constraints into equivalent, but more efficiently evaluated, formulas is another answer.

Pruning constraints can be further 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 the operator has just been

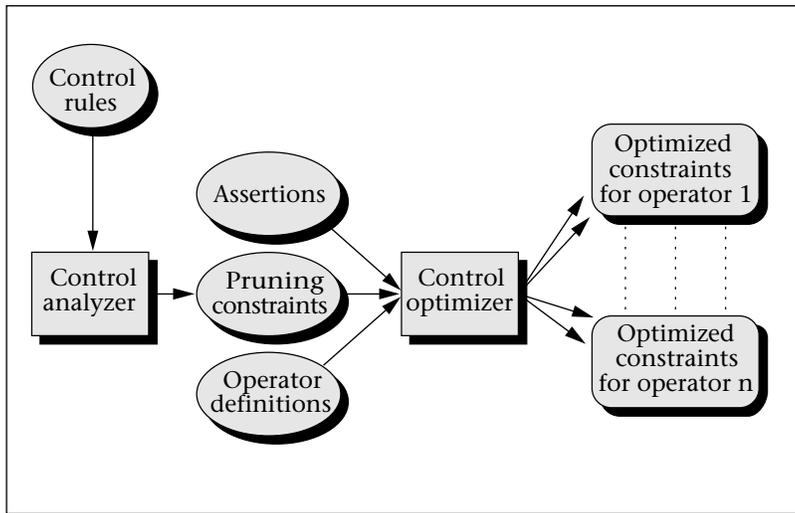


Figure 7. Optimization.

invoked. Although these transformations are done during a preprocessing phase, where exact arguments and time points are not yet known, the operator descriptions contain considerable information about the states in which the pruning constraints will eventually be evaluated.

In many cases, the operator-specific analysis described earlier results in pruning constraints where some conjuncts, or even the entire constraint, 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*.² This control improves performance significantly because control-rule violations can be detected before the planner even attempts to invoke an operator.

In other cases, the process completely removes pruning constraints for certain operators, thus saving the time it takes to evaluate the constraint. This is the case when the rule “do not pick up an object unless it has to be moved” is analyzed relative to the drop and move actions, for example.

TALPLANNER also 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 additional

information sometimes allows TALPLANNER to simplify pruning constraints further, improving the chances of converting constraints into precondition control.

In figure 7, 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 for each operator. The process is fully automated and will be described in detail in a forthcoming paper. Note that the complete preprocessing phase normally takes about 10 to 100 milliseconds. Future versions of TALPLANNER might integrate domain-analysis techniques to automatically generate assertions.

Comparison with TLPLAN

TLPLAN did not compete in the AIPS'00 competition to avoid any conflict of interest because of Fahiem Bacchus's role as organizer of the competition. Although he did a fantastic job organizing and running the competition, TLPLAN's absence was unfortunate because it is one of the fastest planners 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 using the benchmark problems from 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 formula progression techniques. We call this version of TALPLANNER TALPLANNER/PROGRESSION in figures 8 through 11. 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 these figures.

All benchmarks were run on the same 333-megahertz PENTIUM II computer running WINDOWS NT 4.0 SP3, using 256 megabytes of memory. The machine is slow by current standards. For comparison with TLPLAN, we used the pre-compiled version.³ 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).⁴ In all cases, we made sure that the computer was very lightly loaded and that it was never swapping.

Benchmark Results

We used two of the domains chosen for both the AIPS'98 and AIPS'00 competitions: (1) logistics and (2) blocks. All 30 problems from the logistics domain were taken from the AIPS'98 competition. We generated all 43 problems from the blocks domain. 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.

Figure 8 shows how much time the planners needed for the logistics problems, and figure 9 shows how many worlds (states) the planners expanded. TLPLAN creates the first few plans in about a second but needs over 15 hours for some problems and cannot solve 2 of the problems with 256 megabytes 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 megabytes 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 >60,000 seconds, compared to 15.422 seconds for modal TALPLANNER and 0.391 seconds for TALPLANNER using evaluation.

Figure 10 shows how much time the planners needed for the blocks-world problems, and figure 11 shows how many worlds (states) the planners expanded. TLPLAN creates the first few plans in less than 10 seconds but needs almost 9 hours for some problems and cannot solve the larger problems with 256 megabytes 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 >15,000 operators) in under 20 seconds using 63 megabytes of memory. Clearly, 5000 blocks and 15,000 operators is nowhere near the limit. TALPLANNER/EVALUATION with delta states (a more memory-efficient state representation) is somewhat slower, sometimes requiring as much as 26 seconds but only using 27 megabytes of memory. The first 34 problems are solved using less than the 4 megabytes of

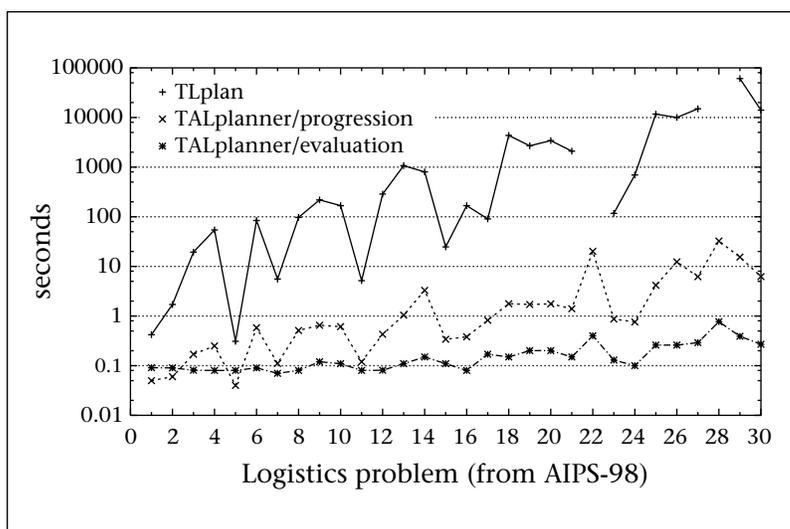


Figure 8. Logistics Problems: Time.

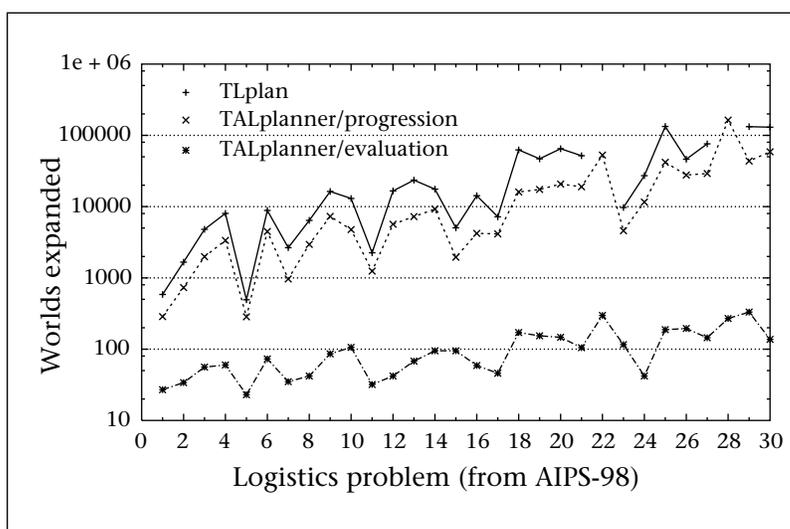


Figure 9. Logistics Problems: States Created.

heap space allocated by the JAVA virtual machine.

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, and Haslum 2000). See Kvarnström and Doherty (2001) for a more detailed description

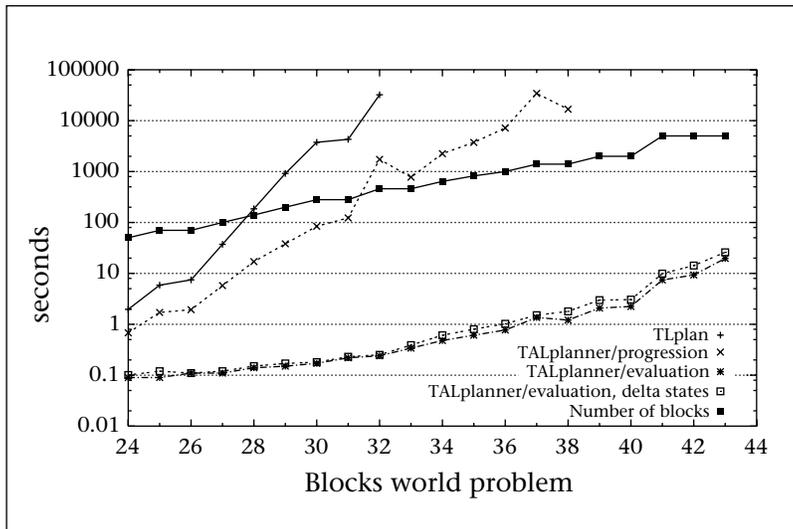


Figure 10. Blocks-World Problems: Time.

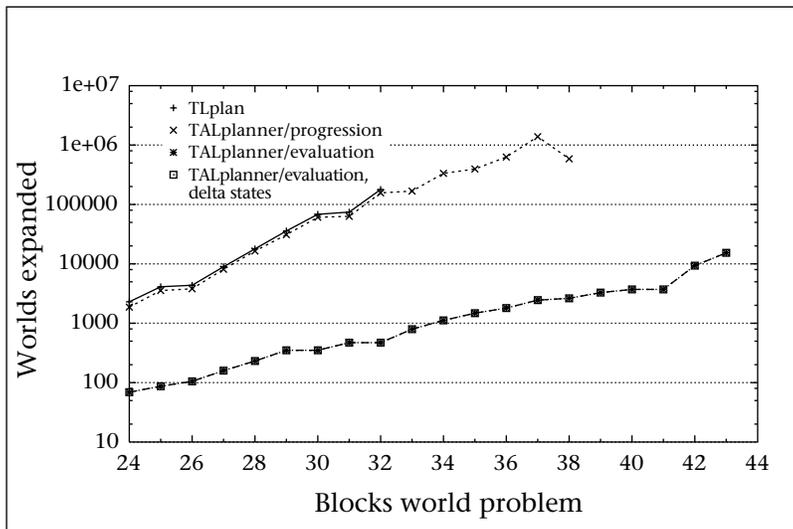


Figure 11. Blocks-World Problems: States Created.

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.

Notes

1. J. Kvarnström and P. Doherty, P. 2000. TALPLANNER project page. Accessible via the KPLAB web page, <http://www.ida.liu.se/~patdo/kplabsite/html/external/>.

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

3. This version can be downloaded from <http://www.lpaig.uwaterloo.ca/~fbacchus/>.

4. Both of these can be downloaded from <http://java.sun.com>.

References

Bacchus, F., and Kabanza, F. 2000. Using Temporal Logics to Express Search Control Knowledge for Planning. *Artificial Intelligence* 116(1-2): 123-191.

Bacchus, F., and Kabanza, F. 1998. Planning for Temporally Extended Goals. *Annals of Mathematics and Artificial Intelligence* 22(1-2): 5-27.

Bacchus, F., and Kabanza, F. 1996. Using Temporal Logic to Control Search in a Forward-Chaining Planner. In *New Directions in AI Planning*, eds. M. Ghallab and A. Milani, 141-153. Amsterdam: IOS Press.

Doherty, P., and Kvarnström, J. 1999. TALPLANNER: An Empirical Investigation of A Temporal Logic-Based Forward-Chaining Planner. In *Proceedings of the Sixth International Workshop on Temporal Representation and Reasoning*, 47-54. New York: IEEE Computer Society Press.

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. 2001. TALPLANNER: A Temporal Logic-Based Forward-Chaining Planner. *Annals of Mathematics and Artificial Intelligence* 30:119-169.

Kvarnström, J.; Doherty, P.; and Haslum, P. 2000. Extending TALPLANNER with Concurrency and Resources. In *Proceedings of the Fourteenth European Conference on Artificial Intelligence*, 501-505. Amsterdam: IOS.

McCarthy, J. 1980. Circumscription—A Form of Non-Monotonic Reasoning. *Artificial Intelligence* 13(1-2): 27-39.

Patrick Doherty is a professor of computer science in the Department of Computer and Information Science (IDA) at Linköping University, Sweden. He is the head of the Artificial Intelligence and Integrated Computer Systems Division at IDA and the Knowledge Processing Laboratory. He is president of the Swedish Artificial Intelligence Society. His current research interests include formal knowledge representation, reasoning about action and change, and deliberative/reactive systems. His e-mail address is patdo@ida.liu.se.

Jonas Kvarnström is a Ph.D. student in the Department of Computer and Information Science at Linköping University, Sweden. He received his M.Sc. in computer science from Linköping University in 1996. His research interests include planning and reasoning about action and change. His e-mail address is jonkv@ida.liu.se.