Anytime Synthetic Projection: Maximizing the Probability of Goal Satisfaction

Mark Drummond[*] and **John Bresina**[†]
Sterling Federal Systems
AI Research Branch, NASA Ames Research Center
Mail Stop: 244-17, Moffett Field, CA 94035

Abstract

This paper presents a projection algorithm for incremental control rule synthesis. The algorithm synthesizes an initial set of goal-achieving control rules using a combination of situation probability and estimated remaining work as a search heuristic. This set of control rules has a certain probability of satisfying the given goal. The probability is incrementally increased by synthesizing additional control rules to handle "error" situations the execution system is likely to encounter when following the initial control rules. By using situation probabilities the algorithm achieves a computationally effective balance between the limited robustness of triangle tables and the absolute robustness of universal plans.

Introduction

We are interested in a continuum of plan-guided systems, from those that can operate entirely off-line, where complete plans are produced in advance and later used by independently competent execution systems, to those systems that are embedded in the situations for which their plans are generated. These embedded systems are especially interesting since they must close the loop between plan formation and plan execution in their environment. For an embedded system, simply generating a plan is not enough; such a system must instead incrementally coerce its environment to conform with its goals. The key tasks for an embedded system are resource-bounded incremental plan synthesis and reactive behavior using appropriate plans in a closed-loop fashion.

The work presented in this paper extends existing theory in the areas of temporal projection, anytime algorithms, and plan synthesis for embedded systems. The goals of this paper are to: 1) define the syntax and semantics of behavioral constraints and provide a search heuristic for their satisfaction; 2) define the probability of behavioral constraint satisfaction; 3) describe a synthetic temporal projection algorithm with anytime properties which heuristically maximizes the probability of behavioral constraint satisfaction.

The next section provides relevant background information. The synthetic temporal projection algorithm is then presented by way of a simple example. The paper concludes with a discussion of connections to related research.

Background

Realistic planning and control problems suggest the need for temporally extended goals of *maintenance* and *prevention*, in addition to the traditional planning goals of *achievement*. Our approach employs a language of *behavioral constraints* which is based on a branching temporal logic (*cf* Drummond, 1989). As an example, consider the following behavioral constraint, or BC.

```
(and
  (prevent (and (drunk driver)
                (has-car-keys driver))
           7 12)
  (achieve (or  (at-home me)
                (have-companion me))
           ?t1))
```

This BC represents a conjunction of two temporally extended goals: the first goal must be false from time 7 through time 12 and the second goal must be true at some arbitrary time in the future. Behavioral constraint semantics are defined in terms of possible behaviors that are synthesized by our temporal projection algorithm. Intuitively, we say that a given projection path w satisfies a behavioral constraint β if and only if all of the formulas in β are true in w over the required time intervals. See appendix A for more details on BC syntax and semantics.

We define a *behavioral constraint strategy* (or BC strategy) to be a partial order over a set of behavioral constraints. The partial order, denoted by "$\prec$",

[*]This work has been partially supported by the Artificial Intelligence Research Program of the Air Force Office of Scientific Research.

[†]Also affiliated with the Computer Science Department at Rutgers University.

indicates both execution and problem solving precedence. Behavioral constraint strategies for a given behavioral constraint are produced using domain- and problem-specific planning expertise. The BC strategy constructed for a given BC indicates a set of subproblems for the projector to satisfy and an order in which to satisfy them. This process is beyond the scope of this paper; please refer to Bresina and Drummond (1990) for more information. The way in which BC strategies are used by the projector is made clear in the next section.

In order to project future possible courses of action our projector needs a *causal theory* for each domain of application. A causal theory is a set of operators which defines both the *actions* that the system can take and the exogenous *events* that can occur in the application environment. The difference between actions and events is simply this: actions can be chosen for execution by the control system under construction (*e.g.*, move in a direction) while the occurrence of events is determined by the system's environment (*e.g.*, a gust of wind). From the perspective of the projector however, actions and events are similar, and both can be characterized as a situation to situation transition.

The projector explores various possible futures by repeatedly finding enabled operators and applying them to produce new hypothetical situations. The projector creates a directed acyclic graph, where each node denotes a domain situation and each arc is labelled with a domain operator. Projection associates a duration with each operator application and uses this to calculate a time stamp for the resulting situation.

A path in a projection graph denotes a future possible behavior. Projection paths which satisfy a given behavioral constraint are compiled into a set of Situated Control Rules (SCRs) similar to the way that a STRIPS plan is transformed into a triangle table (Fikes *et al.*, 1972). The SCRs indicate to the reaction component those actions which will "lead to" the eventual satisfaction of its current behavioral constraint. SCRs are used by the reaction component as a set of local instructions constituting a control program.

See Bresina and Drummond (1990) and Drummond (1989) for more details about our overall architecture. The algorithm described in this paper does not critically depend on the architecture, so many irrelevant details have been suppressed. Our temporal projection algorithm can be used by a variety of systems, in a range of architectures.

The Projection Algorithm

This section presents our anytime synthetic projection algorithm. We start with a description of the algorithm in operation and then present ways to control the search that is inherent in this approach.

The *project* algorithm accepts a behavioral constraint and domain causal theory; it attempts to maximize the probability that the reaction component will satisfy the behavioral constraint. Our algorithm is based on the heuristic search paradigm which makes it hard to guarantee that the actual *maximum* probability will be found. Instead, as is typically done with heuristic search algorithms, we claim only that our algorithm attempts to maximize the probability of goal satisfaction, which we refer to as *heuristic maximization.*

To simplify the presentation we characterize the projector's causal theory as a single function called *transition.* The function *transition* (s) maps a situation description s to a set of triples $< s_i, p_i, o_i >$ such that $p(s_i \mid s, o_i) = p_i$, where the conditional probability expression has the following interpretation. If o_i denotes an action, then p_i is the probability that s_i will be the resulting situation if o_i is executed in situation s. If o_i denotes an event, then p_i is the probability that s_i will be the resulting situation if o_i occurs in situation s. For a given s, we assume that the possible transitions are mutually exclusive. Notice that this definition of *transition*(s) makes the Markov assumption by ignoring the particular sequence of operators used to produce s. It is difficult to achieve a complete specification of all possible situation transitions in a realistic domain, and the automatic incremental improvement of the transition function specification is part of our future research agenda.

See figure 1 for an abstract projection graph example. The behavioral constraint strategy $\beta_1 \prec \beta_2$ has been selected as an appropriate way to satisfy β. This BC strategy indicates that a path which satisfies β_1 composed with a path which satisfies β_2 will constitute a path which satisfies β.

Project first calls *traverse* to find a single path that satisfies $\beta_1 \prec \beta_2$ from its "current" situation, s_1. *Traverse* uses the function *transition* to create situations reachable under the application of a single operator from s_1. Not all possible transitions are considered: a filter is used to select a subset of the most probable transitions, and only these are used to produce new successors to s_1. In our example only s_2 survives the probability filter. The number of survivors under this winnowing operation is determined by a filter-width parameter, corresponding to the filter selection function in Ow and Morton's (1986) *filtered beam search.*

A heuristic value is calculated for each successor situation based on the situation's probability and an estimate of the remaining work required to satisfy β_1 from that situation. (This estimation function is explained in more detail below.) Another winnowing process is used to select a subset of these situations that have the highest heuristic value. For our example, this set contains only s_2. In general, however, this set will contain a subset of all possible frontier search nodes in the developing projection graph. The number of elements in this set is limited by a beam-width parameter, corresponding to Ow and Morton's (1986) beam selection function. This set of frontier nodes is passed on to a

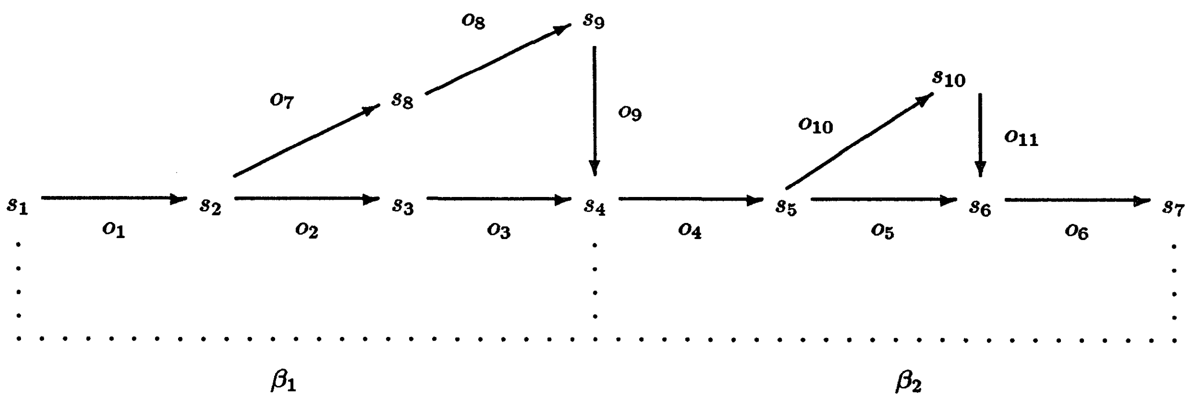

Figure 1: A Simple Projection Graph Example

recursive call of *traverse.*

Traverse continues to extend projection paths by selecting possible transitions until it finds a path which satisfies β_1. In the figure, the first satisfactory path discovered is $s_1 o_1 s_2 o_2 s_3 o_3 s_4$. Situated Control Rules are now compiled for each situation in this path. The reaction component will thus be given a set of rules of the form: IF s_i AND β_1 THEN o_i, for $i = 1, 2, 3$. At this point, *traverse* focuses its search for a solution path to β_2 in the subspace anchored at s_4. This is accomplished by collapsing the set of frontier nodes to the singleton set $\{s_4\}$. Such a collapse has the effect of requiring any solution path for β_2 to start in the situation terminating the satisfactory path for β_1. Winnowing the set of possibilities in this way helps to control the projector's search by reducing the number of alternative situations in the expanding search frontier.

We call this strategy *cut-and-commit*, and it is one aspect of the algorithm's anytime operation. The conditions under which this approach is advisable are discussed below.

Traverse continues its search to satisfy β by finding a projection path which satisfies β_2 from s_4. In our figure, the eventual satisfactory path for β_2 is $s_4 o_4 s_5 o_5 s_6 o_6 s_7$. This path is passed to the SCR compiler producing another set of SCRs for the satisfaction of β_2. The probability for the path s_1 through s_7 can be calculated from $p(s_{i+1}|s_i, o_i)$, $i = 1, \ldots, 6$ (as defined below). This number gives us a lower bound on the probability that the reaction component will satisfy β. Assuming that there is still time before the reaction component must take action, we can increase this probability by finding additional paths which also satisfy β. Each additional path will serve to increase the lower bound on the reaction component's probability of satisfying β.

Robustify is our algorithm for finding additional projection paths. The algorithm finds high-probability deviations from the single existing solution path and calls *traverse* to find alternative paths which recover from each deviation. A deviation is a transition in a situation which produces a new situation from which there does not yet exist a satisfactory path. For example, when *robustify* is applied to the path $s_1 o_1 s_2 o_2 s_3 o_3 s_4$, it finds that the transition to s_8 via operator o_7 has a high probability of occurring in s_2. *Traverse* is used to recover from this deviation by synthesizing an alternate path, $s_1 o_1 s_2 o_7 s_8 o_8 s_9 o_9 s_4$, which also satisfies β_1. Similarly, *robustify* finds that the transition from s_5 to s_{10} via o_{10} has high probability and calls *traverse* to synthesize the path $s_4 o_4 s_5 o_{10} s_{10} o_{11} s_6 o_6 s_7$. Each additional path serves to increase the probability that β will be satisfied by increasing the probability that each of its component constraints, β_1 and β_2, will be satisfied.

Situated Control Rules are compiled for each new subpath synthesized by *traverse*; in our example, new SCRs are created for s_2, s_8, s_9, s_5, and s_{10}. This incremental *deviate-and-recover* strategy is another aspect of the algorithm's anytime operation. As each new path is found, SCRs are given to the reaction component to help it deal with ever more of the possible domain situations in which it might find itself.

Controlling the Search

Situation probability and estimated remaining work were used in *traverse* to define a heuristic evaluation function. The heuristic value for a situation s, with respect to a BC β, is computed as: $h(s, \beta) = \text{K1} \cdot p(s) + \text{K2} \cdot rwt(s, \beta)$, where $p(s)$ is the probability of situation s and $rwt(s, \beta)$ is the estimated remaining work required to satisfy β from s. The user-provided weights, K1 and K2, determine the relative importance of low-cost and high-probability in the computation of h and, hence, affect the type of solutions synthesized by *traverse*. These parameters must be tuned as required for each domain of application. This section gives def-

$$
\begin{aligned}
rw(s,\ (and\,\beta_1\,\ldots\,\beta_n)) &= \textstyle\sum_{i=1}^{n} rw(s,\beta_i)\\
rw(s,\ (or\,\beta_1\,\ldots\,\beta_n)) &= \min_{i=1}^{n} rw(s,\beta_i)\\
rw(s,\ (maintain\ \psi\ \tau_s\ \tau_e)) &= \text{KW}\cdot \textit{min-true}\ (\psi,s)\cdot(\tau_e-\tau_n) + \textit{min-false}\ (\psi,s)\cdot(\text{CW}+\text{KW}\cdot(\tau_e-\tau_s))\\
rw(s,\ (prevent\ \psi\ \tau_s\ \tau_e)) &= \text{KW}\cdot \textit{min-true}\ (\psi,s)\cdot(\tau_e-\tau_n) + \textit{min-false}\ (\psi,s)\cdot(\text{CW}+\text{KW}\cdot(\tau_e-\tau_s))\\
rw(s,\ (maintain\ \psi\ \varphi\ \varphi)) &= \text{KW}\cdot \textit{min-true}\ (\psi,s) + \text{CW}\cdot \textit{min-false}\ (\psi,s)\\
rw(s,\ (prevent\ \psi\ \varphi\ \varphi)) &= \text{KW}\cdot \textit{min-true}\ (\psi,s) + \text{CW}\cdot \textit{min-false}\ (\psi,s)
\end{aligned}
$$

Table 1: Definition of $rw(s,\beta)$

initions for estimated remaining work and path probability, and more clearly explains the role of behavioral constraint strategies in controlling search.

Estimated remaining work

For planners concerned only with conjunctive goals of achievement, a heuristic based on situation difference gives reasonable results (Nilsson, 1980); to handle behavioral constraints we have generalized the notion of situation difference to that of *remaining work per time.*

Our heuristic uses two global parameters, KW (*k*eep *w*ork) and CW (change *w*ork), which relate predicate truth value to work. The parameter KW denotes the minimum work per unit time to keep the truth value of a predicate constant. The parameter CW denotes the minimum number of work units required to change the truth value of a predicate. We assume that facts change instantaneously and CW estimates the minimum work required to change the truth value of a randomly selected predicate. A user must set these parameters as required for each application domain.

We define the remaining work per time $rwt(s,\beta)$ to be $rw(s,\beta)/rt(s,\beta)$; where $rw(s,\beta)$ is the remaining work necessary to satisfy β from s and $rt(s,\beta)$ is the remaining time in which to do the work. The remaining time can be easily estimated from β and s. Let s be a situation, and let τ_n be the time stamp of s. The numerator of our equation, $rw(s,\beta)$, can then be defined as shown in table 1.

The function *min-true*(ψ,s) gives the minimum number of predicates in the formula ψ that are true in situation s. Similarly, *min-false*(ψ,s) gives the minimum number of false predicates. These terms, together with CW and KW, produce an optimistic estimate of the amount of remaining work.

For example, consider the evaluation of $rw(s,$ $(maintain\ \psi\ \ \tau_s\ \tau_e))$. The formula ψ must be maintained from time point τ_s through time point τ_e, from situation s with time stamp τ_n. The appropriate definition in table 1 has two terms: the first term describes the work required to keep the minimum number of true predicates in ψ true from τ_n through τ_e; the second term deals with the work required to change the minimum number of false predicates in ψ to be true, and the work required to keep these predicates true from τ_s through τ_e. The classical situation difference heuristic is a degenerate form of these measures, where work is measured in the number of predicates that must be made true and where there is no cost for keeping predicates true over time.

Goal Satisfaction Probability

Our description of *traverse* depended on the ability to combine individual transition probabilities into aggregate projection path probabilities; this section explains how this is accomplished.

Let $G = (S,T)$ be a projection graph, where S is a set of possible situations and T is a set of situation-to-situation transitions; let $s \in S$ be a particular situation, and let $w = s_1 o_1 s_2 o_2 \ldots o_{n-1} s_n$ be a path in G. The *path probability* of w is defined to be the product of the transition probabilities in w: $p\,(w) = p\,(s_1)\cdot \Pi_{i=1}^{n-1} p\,(s_{i+1} \mid s_i, o_i)$.

For a situation s, the *situation probability* is defined as the sum of the path probabilities of all paths from the unique starting situation of G, ss, to s: $p\,(s) = \sum p\,(w)$ summed over $\{w : w = s_1 o_1 \ldots o_{n-1} s_n$ is a path in G, $s_1 = ss$, and $s_n = s\}$.

Finally, we can define the probability that a behavioral constraint, β, is satisfied by a projection graph, G, as the sum of the probabilities of all paths in G anchored at the unique starting situation ss which satisfy β: $p\ (\beta \mid G) = \sum p\ (w)$ summed over $\{w : w = s_1 o_1 \ldots o_{n-1} s_n$ is a path in G, $s_1 = ss$, and w satisfies $\beta\}$.

The probability that the reaction component will satisfy a BC β under the guidance of the SCRs compiled from a projection graph G is bounded below by $\mathrm{p}\ (\beta \mid G)$. The probability $\mathrm{p}\ (\beta \mid G)$ is a lower bound because the reaction component might have access to other SCRs relevant to β which cover situations that are not in G.

Behavioral Constraint Strategies

As mentioned above, a behavioral constraint strategy is a partial order over a set of behavioral constraints. A given BC strategy controls search by giving the projector a set of behavioral constraints to satisfy and an order in which to satisfy them. A BC strategy is satisfied when each of its component constraints is satisfied in an order consistent with the given partial order.

To make this idea more precise, let $(\Gamma, \prec)$ be a BC strategy, where Γ contains n behavioral constraints; let Θ be the set of all total orders over Γ compatible with

$\prec$. The objective for *traverse* is to synthesize a path $w = w^1 \circ w^2 \circ \cdots \circ w^n$, such that there exists a total order $\theta \in \Theta$ where for each w^i, w^{i+1} in w, there exists $\beta \prec \beta' \in \theta$ such that w^i satisfies β and w^{i+1} satisfies β'. Furthermore, each $\beta \in \Gamma$ must be satisfied by one w^i in w. The "$\circ$" operator represents path composition defined as follows: $w \circ w' = s_1 o_1 s_2 o_2 \ldots s_i e'_1 s'_2 e'_2 \ldots s'_j$, where $w = s_1 o_1 s_2 o_2 \ldots s_i$ and $w' = s'_1 e'_1 s'_2 e'_2 \ldots s'_j$, if the union of s_i and s'_1 is consistent, else $w \circ w'$ is undefined.

Consider the simple example used above where the BC strategy is $\beta_1 \prec \beta_2$. In the ideal case, for each path w^1 that satisfies β_1, there exists a path w^2 that satisfies β_2 such that $w^1 \circ w^2$. In this case, our cut-and-commit strategy will never be forced to backtrack over the first solution found for β_1, and the policy of immediate SCR compilation is risk-free. However, it is not always possible to construct such ideal BC strategies. More typically only a subset of the paths which satisfy β_1 can be extended to also satisfy β_2. In this case, the projector might have to backtrack to find another solution to β_1. If such backtracking occurs, then (at least some of) the SCRs that were compiled from a rejected solution to β_1 are not appropriate in the context of $\beta_1 \prec \beta_2$. However, they may be appropriate in the context of another BC strategy and hence could still prove useful.

In this paper, we do not address what the reactor does when more than one SCR is applicable. This issue is part of our current research effort; we are developing a SCR conflict resolution strategy based on the BC strategy context for which an SCR is appropriate in combination with the transition probability and the remaining work estimates associated with an SCR. In our ongoing research on the interaction between the projector and the automatic production of behavioral constraint strategies, one future topic will be techniques for assessing and reducing the risk of backtracking over the inter-behavioral constraint "cut" points.

Discussion

A triangle table (Fikes *et al.*, 1972) is analogous to what you get after running *traverse* only once, a universal plan (Schoppers, 1987) is analogous to what you get by doing exhaustive search of the space of possible domain situations. A triangle table is like a set of SCRs designed to deal with each situation in a sequence of situations, and a universal plan is like a set of SCRs which has 100% coverage of the space of situations. Ginsberg (1989) has argued against the practicality of universal plans. He has suggested that for "cognitive tasks", a system should be able to enhance its performance by expending additional mental resources. Our projection algorithm does exactly this. Under our approach, additional computation time serves to increase the probability of goal satisfaction.

There are various architectures addressing the real-time embedded control problem. Representative approaches include Brooks' (1985) *subsumption architecture*, Nilsson's *action nets* (Nilsson, *et al.*, 1990), Maes' (1990) spreading activation approach, and the *situated automata* of Rosenschein and Kaelbling (Rosenschein, 1989; Rosenschein & Kaelbling 1986; Kaelbling, 1987a,b, 1988). Each of these approaches gives a designer a language and methodology for specifying a control system.

Brooks' (1985) *subsumption architecture* provides an elegant way of organizing the functional components of an embedded control system. The subsumption architecture "model" of embedded execution is richer than our simple IF–THEN Situated Control Rule view. However, we are able to synthesize SCRs automatically from a given behavioral constraint and causal theory describing a particular application domain. To our knowledge, Brooks has not yet addressed the automatic synthesis of subsumption architecture instances.

Nilsson's *action nets* (Nilsson, *et al.*, 1990) provide another methodology and language for the description of embedded systems. Nilsson's view of closed-loop homeostatic servo mechanisms is appealing, and early results are promising. Our work differs in providing a more expressive language of behavioral constraints and by using information about situation probability to control search.

Maes' (1990) system employs a spreading activation approach for dynamic action selection and can be viewed as a form of on-line action synthesis. The behavior of Maes' algorithm depends on a number of global parameters which are set by the user based on (among other factors) characteristics of the environment and the specific goal to be achieved. Hence, if the nature of the environment changes or if the desired goal changes, the user will need to re-tune the parameters. Our work differs by explicitly searching through the space of possible futures. A behavioral constraint is one of the algorithm's inputs; hence, changes in the system's goals are taken into account automatically. Changes in the nature of the environment would be reflected in the transition probabilities; hence, updated probabilities would appropriately influence the projection search.[1]

The most closely related work is that of Rosenschein and Kaelbling (Rosenschein, 1989; Rosenschein & Kaelbling 1986; Kaelbling, 1987a,b, 1988). Kaelbling's GAPPS system is a compiler which translates goal reduction expressions into directly executable circuits. However, a person writing GAPPS goal reductions must essentially do their own temporal projection; that is, it is the person's responsibility to guarantee that the rules, once sequenced, will "lead to" goal satisfaction. In contrast, our approach defines a temporal projection mechanism which sorts out the effects

[1]We have not yet implemented the automatic update of transition probabilities.

of various action sequences automatically. Of course, we potentially pay a greater computational cost by carrying out this search. Additionally, the GAPPS system, and the REX language on which it is based, have a great deal to say about bounded reaction time in terms of the circuits synthesized from higher-level expressions. We are not currently addressing this issue.

We stress the *synthetic* nature of our projector to distinguish it from *analytic* projection (Dean & McDermott, 1987; Hanks, 1990). An analytic projector is used by a planner to validate plans while a synthetic projector combines operator selection and validation in the same algorithm. The analytic/synthetic distinction is largely one of perspective, since it is possible to view a planner–analytic projector pair as a complete system which performs synthetic projection.

Hanks (1990) greatly extended the capabilities of temporal projection systems by adding information regarding probability. Dean and Kanazawa (1988) also use similar information. The techniques of Hanks, Dean and Kanazawa can be used to judge the probability that a given fact will be true at an arbitrary point in the future. We can imagine providing such an inferential facility, but for now, we permit only calculations of individual situation probability. The algorithms of Hanks, Dean and Kanazawa can perform more powerful inferences.

Dean and Boddy (1988) have characterized an *anytime algorithm* as one which can be asked for an answer at any point, where the algorithm's answers are expected to improve the longer it is allowed to run. Our use of *traverse* and *robustify* satisfy this characterization, in the sense that a set of SCRs is available for the reactor at any point in time, and in the sense that the set of SCRs "improves" over time by incrementally increasing goal satisfaction probability. We have identified two ways in which a synthetic temporal projection algorithm can be considered "anytime": first, by using our cut-and-commit search strategy based on behavioral constraint strategies; and second, by recursively employing our deviate-and-recover strategy to manage probable errors.

Our cut-and-commit approach ameliorates the complexity of the projection search. To see this, suppose that the average branching factor in the projection is b, and suppose that an eventual solution path is of length n. This means that breadth-first search would have to project, in the worst case, $b^{n+1} - 2$ many situations to find a successful path. Suppose that the projector's BC strategy is totally ordered and is of length c. In the worst case, the number of situations that *traverse* must project is $c \cdot b^{(n/c)+1} - 2c$. As c approaches n, the number of situations we must consider falls off dramatically. This assumes, of course, that no backtracking occurs. As c increases, the projection takes on the shape of a series of small trees connected end-to-end, rather than one large tree running from start to finish. The larger c is, the smaller the computation's anytime "grain size" becomes.

We have designed and implemented a simulator for an experimental domain called the Reactive Tile World. The Reactive Tile World exhibits exogenous events and temporally extended goals of maintenance and prevention. We are in the process of empirically validating our projection algorithm on a suite of Reactive Tile World test problems.

Acknowledgements

Other members of the ERE group, namely, Rich Levinson, Andy Philips, Nancy Sliwa, and Keith Swanson have helped us develop these ideas; discussions with Mark Boddy, Leslie Kaelbling, Stan Rosenschein, and Steve Hanks have been useful. Thanks to John Allen, Hamid Berenji, Guy Boy, Peter Cheeseman, Smadar Kedar, Phil Laird, and Amy Lansky for useful comments on a previous draft. Final responsibility for all errors and omissions rests, of course, with the authors. Thanks also to Peter Friedland for providing an excellent research environment at NASA Ames.

Appendix A: Behavioral Constraint Syntax and Semantics

A behavioral constraint (BC) is an expression constructed according to the following grammar. We use the symbol β to stand for an arbitrary BC and the symbol | to indicate alternatives.

$$\begin{array}{lll}
\beta & \rightarrow & (\textit{and}\ \beta_1\,\beta_2\ \ldots\ \beta_n)\ \mid\ (\textit{or}\ \beta_1\,\beta_2\ \ldots\ \beta_n) \\
\beta & \rightarrow & (\textit{maintain}\ \psi\ \tau_1\ \tau_2)\ \mid\ (\textit{prevent}\ \psi\ \tau_1\ \tau_2) \\
\beta & \rightarrow & (\textit{maintain}\ \psi\ \varphi\ \varphi)\ \mid\ (\textit{prevent}\ \psi\ \varphi\ \varphi) \\
\psi & \rightarrow & (\textit{and}\ \psi_1\,\psi_2 \ldots\ \psi_n)\ \mid\ (\textit{or}\ \psi_1\,\psi_2 \ldots\ \psi_n) \\
\psi & \rightarrow & \textit{predicate}
\end{array}$$

We use ψ to denote a formula, τ to denote a time point constant, and φ to denote a time point variable. Time points are natural numbers. A variable is indicated by a question-mark, for instance: $?t$. All variables are implicitly existentially quantified. We currently use time point variables only to express those goals of "achievement" or "destruction" which are not required to occur at a predetermined point in time; these goals are given the following syntactic forms: $(\textit{maintain}\ \psi\ \varphi\ \varphi) \equiv (\textit{achieve}\ \psi\ \varphi)$ and $(\textit{prevent}\ \psi\ \varphi\ \varphi) \equiv (\textit{destroy}\ \psi\ \varphi)$.

Behavioral constraint semantics are defined in terms of projection graph paths. Let $w = s_1 o_1 s_2 o_2 \ldots o_{n-1} s_n$ be a projection graph path; let $ts\,(s)$ denote the time stamp of situation s; and let β be a behavioral constraint. Then w satisfies β under the following conditions.

$$\begin{array}{lll}
w & \models & (\mathit{and}\ \beta_1 \ldots \beta_n) \\
 & \text{iff} & \forall i \in \{1 \ldots n\} :\ w \models \beta_i \\
w & \models & (\mathit{or}\ \beta_1 \ldots \beta_n) \\
 & \text{iff} & \exists i \in \{1 \ldots n\} :\ w \models \beta_i \\
w & \models & (\mathit{maintain}\ \psi\ \tau_1\ \tau_2) \\
 & \text{iff} & \exists s_i \in w :\ ts(s_i) \leq \tau_1 \text{ and } s_i \models \psi \\
 & & \quad \text{and } \forall s_j \in w, j > i : \\
 & & \qquad s_j \models \psi \text{ or } ts(s_j) > \tau_2 \\
w & \models & (\mathit{prevent}\ \psi\ \tau_1\ \tau_2) \\
 & \text{iff} & \exists s_i \in w :\ ts(s_i) \leq \tau_1 \text{ and } s_i \not\models \psi \\
 & & \quad \text{and } \forall s_j \in w, j > i : \\
 & & \qquad s_j \not\models \psi \text{ or } ts(s_j) > \tau_2 \\
w & \models & (\mathit{maintain}\ \psi\ \varphi\ \varphi) \\
 & \text{iff} & \exists s_i \in w :\ s_i \models \psi \\
w & \models & (\mathit{prevent}\ \psi\ \varphi\ \varphi) \\
 & \text{iff} & \exists s_i \in w :\ s_i \not\models \psi \\
s & \models & (\mathit{and}\ \psi_1 \ldots \psi_n) \\
 & \text{iff} & \forall i \in \{1 \ldots n\} :\ s \models \psi_i \\
s & \models & (\mathit{or}\ \psi_1 \ldots \psi_n) \\
 & \text{iff} & \exists i \in \{1 \ldots n\} :\ s \models \psi_i \\
s & \models & \mathit{predicate} \\
 & \text{iff} & \mathit{predicate} \in s
\end{array}$$

References

[1] Bresina, J., and Drummond, M. 1990. Integrating Planning and Reaction: A Preliminary Report. *Proceedings of the 1990 AAAI Spring Symposium Series* (session on Planning in Uncertain, Unpredictable, or Changing Environments).

[2] Bresina, J., Marsella, S., and Schmidt, C. 1986. REAPPR – Improving Planning Efficiency via Expertise and Reformulation. Rept. LCSR-TR-82, LCSR, Rutgers University, June.

[3] Brooks, R. 1985. A Robust Layered Control System for a Mobile Robot. Technical Report 864, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts.

[4] Dean, T., and Boddy, M. 1988. An Analysis of Time-Dependent Planning. *AAAI-88*. pp. 49–54.

[5] Dean, T., and Kanazawa, K. 1989. A Model for Projection and Action. *Proceedings of IJCAI-89*. pp. 985–990.

[6] Dean, T., and McDermott, D. 1987. Temporal Database Management. *AI Journal*. Vol. 32(1). pp. 1–55.

[7] Drummond, M. 1989. Situated Control Rules. *Proceedings of Conference on Principles of Knowledge Representation & Reasoning*. Toronto, Canada.

[8] Ginsberg, M. 1989. Universal Planning: An (Almost) Universally Bad Idea. *AI Magazine*, Vol. 10, No. 4. pp. 40–44.

[9] Hanks, S. 1990. Projecting Plans for Uncertain Worlds. Yale University, CS Department, YALE/CSD/RR#756.

[10] Fikes, R., Hart, P., and Nilsson, N. 1972. Learning and Executing Generalized Robot Plans. *AI Journal*, Vol 3, pp. 251–288.

[11] Fikes, R. and Nilsson. N. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *AI Journal*, Vol. 2, pp. 189–208.

[12] Kaelbling, L. 1987a. An Architecture for Intelligent Reactive Systems. *Reasoning About Actions and Plans.* M. Georgeff and A. Lansky, Eds., Morgan Kauffman.

[13] Kaelbling, L. 1988. Goals as Parallel Program Specifications. *Proceedings of the Seventh National Conference on Artificial Intelligence.* St. Paul, Minnesota.

[14] Maes, P. 1990. How To Do the Right Thing. *Connection Science Journal.* (Special Issue on Hybrid Systems. J. Hendler, editor).

[15] Nilsson, N., Moore, R., and Torrance, M., ACT-NET: An Action Network Language and its Interpreter. Draft paper, Stanford Computer Science Department, February 1990.

[16] Nilsson, N. 1980. *Principles of Artificial Intelligence.* Tioga Publishing Company, CA.

[17] Ow, P. and Morton, T. 1986. Filtered Beam Search in Scheduling. Working paper, Graduate School of Industrial Administration, Carnegie-Melon University.

[18] Rosenschein, S. 1989. Synthesizing Information-Tracking Automata from Environment Descriptions. *Proceedings of Conference on Principles of Knowledge Representation & Reasoning.* Toronto, Canada.

[19] Rosenschein, S. and Kaelbling, L. 1986. The Synthesis of Digital Machines with Provable Epistemic Properties. *Proceedings of Workshop on Theoretical Aspects of Knowledge.* Monterey, CA (March 13-14).

[20] Schoppers, M. 1987. Universal Plans for Reactive Robots in Unpredictable Environments. *Proceedings of the Tenth International Conference on Artificial Intelligence.* pp. 1039–1046, Milan, Italy.