Temporal Planning with Continuous Change*

J. Scott Penberthy
IBM T.J. Watson Research Center
30 Saw Mill River Road
Hawthorne, NY 10532
jsp@watson.ibm.com

Daniel S. Weld
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98105
weld@cs.washington.edu

Abstract

We present ZENO, a least commitment planner that handles actions occurring over extended intervals of time. Deadline goals, metric preconditions, metric effects, and continuous change are supported. Simultaneous actions are allowed when their effects do not interfere. Unlike most planners that deal with complex languages, the ZENO planning algorithm is sound and complete. The running code is a complete implementation of the formal algorithm, capable of solving simple problems (*i.e.*, those involving less than a dozen steps).

Introduction

We have built a least commitment planner, ZENO, that handles actions occuring over extended intervals of time and whose preconditions and effects can be temporally quantified. These capabilities enable ZENO to reason about deadline goals, piecewise-linear continuous change, external events and to a limited extent, simultaneous actions. While other planners exist with some of these features, ZENO is different because it is both sound and complete.

As an example of ZENO's capabilities, consider a toy world in which a single plane moves passengers between cities. "Slow flying" travels at 400 miles per hour and consumes 1 gallon of fuel every 3 miles, on average. "Fast flying" travels at 600 miles per hour and consumes 1 gallon of fuel every 2 miles. Passengers can be boarded in 30 minutes and deplaned in 20 minutes. Refueling gradually increases the fuel level to a maximum of 750 gallons, taking one hour from an empty tank. Boarding, deplaning, and refueling must all occur while the plane is on the ground. The plane flies routes between 4 cities as shown in figure 1.

Suppose that **dan** and **ernie** are at **city-c**, but the empty plane and **scott** are at **city-a**. If the plane only has 500 gallons of fuel, how can we ensure that **scott** and **ernie** get to **city-d** in less than $5 + \frac{1}{2}$ hours?

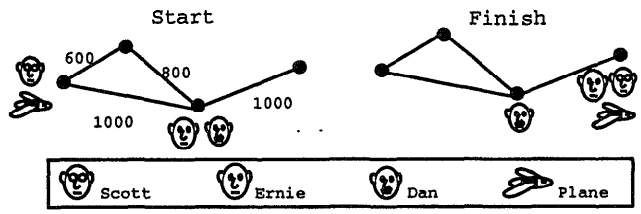
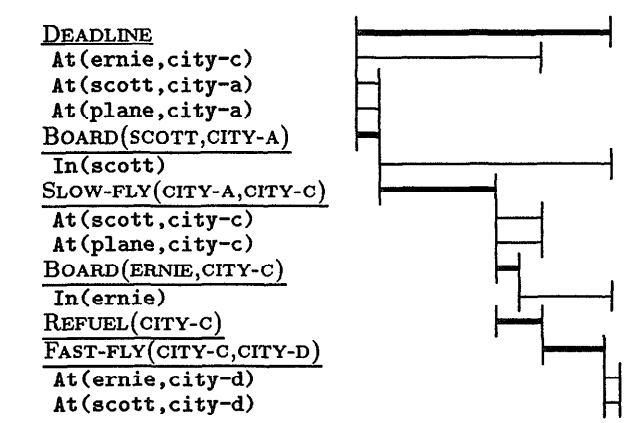


Figure 1: Airplane routes and a sample problem

Synthesizing the solution requires reasoning about simultaneous actions and continuous change. Since the plane's fuel diminishes at differing rates depending on the speed of flight, ZENO needs to trade off speed for efficiency. The planner must also handle conditional effects, since passengers are moved only when they are aboard. Deadline goals are present: the plane must meet a tight schedule. ZENO takes about three minutes to solve the problem. The Gantt chart below depicts ZENO's plan — facts established by actions are shown as indented formulae, and the bars to the right indicate the intervals of time over which actions occur or facts persist.



The constraints (over 100 of them) can be para-

*This research was funded in part by the IBM Corporation, National Science Foundation Grant IRI-8957302, Office of Naval Research Grant 90-J-1904 and a grant from the Xerox corporation.

phrased as follows. Passenger **scott** takes 30 minutes to board the plane at **city-a**. Since it doesn't have enough fuel to **fast-fly**, the plane flies slowly from **city-a** to **city-c**, taking 2 hours, 30 minutes. The plane is refueled over the next hour at **city-c**; meanwhile passenger **ernie** climbs onboard. Note that the deadline could not be met without scheduling these actions simultaneously. Finally, over the next 1:20, the plane is flown quickly from **city-c** to **city-d**. This leaves 235 gallons of fuel in the plane for a total plan time of 5 hours, 20 minutes — 10 minutes to spare. The rest of this paper describes our action and plan representation, provides an overview of the ZENO algorithm, then discusses formal and empirical aspects.

Actions and goals

ZENO uses a typed, first-order language with equality to describe goals and the effects of actions. A point-based model of time is adopted; temporal functions and relations use a time point as their first argument. Quantifiers specify the type of the quantified variable (*i.e.*, $\forall_{\mathbf{type}}$ and $\exists_{\mathbf{type}}$). All types except **time** are assumed finite. To represent maintenance (*i.e.*, interval) goals and piecewise-linear continuous effects, one simply specifies universal quantification over variables of type **time**.

A ZENO action schema (*e.g.*, figure 2) characterizes a set of possible actions with sentences from ZENO's

Schema Fast-Fly $(m,\ l)$
at-time: $[t_s, t_e]$
precondition:
$\forall_{\mathbf{time}}\, t\ t \in [t_s, t_e] \supset$ $fuel(\mathrm{t}, plane) > 0$ $\wedge$
$\mathbf{at}(t_s, \mathbf{plane}, m)$ $\wedge$
$\mathbf{dist}(m,l)=\nu_2 \wedge \mathbf{mpg}(\mathbf{plane})=\nu_3$
constraints:
$\nu_4 = -600/\nu_3,\ t_e = t_s + \nu_2/600$
effect:
$\mathbf{at}(t_e, \mathbf{plane}, l)$ $\wedge$
$\forall_{\mathbf{time}}\, t\ t \in (t_s, t_e] \supset \neg\mathbf{at}(t, \mathbf{plane}, m)$ $\wedge$
$[\forall_{\mathbf{human}}\, o\ \forall_{\mathbf{time}}\, t$
$(t \in (t_s, t_e] \wedge \mathbf{in}(t, o)) \supset \neg\mathbf{at}(t, o, m) \wedge \mathbf{at}(t_e, o, l)] \wedge$
$\forall_{\mathbf{time}}\, t\ t \in [t_s, t_e] \supset \frac{\partial}{\partial t}\mathbf{fuel}(t, \mathbf{plane}) = \nu_4$

Figure 2: An action schema for fast flying.

logic. A schema specifies the time over which an action occurs, the preconditions for execution, and the effects on the world. For example, the precondition $\mathbf{at}(t_s, \mathbf{plane}, m)$ insists that the plane start at location m at time t_s in order to fly from location m to location l. The first precondition of **Fast-Fly**,

$$\forall_{\mathbf{time}}\, t\ t \in [t_s, t_e] \supset \mathbf{fuel}(t, \mathbf{plane}) > 0$$

restricts the plane's fuel level to remain above zero while flying, but does not commit to a single value: $\mathbf{fuel}(t,\mathbf{plane})$ may vary throughout the interval $[t_s, t_e]$. While a goal may be any quantified sentence composed of logical connectives ($\wedge$, $\vee$) and literals ($f(t, x_1 \ldots x_n) = c$, $R(x_1 \ldots x_n)$, or $\neg R(x_1 \ldots x_n)$), disjunction and existential quantification are banned from action effects. Internally, the conjunctive effects of each action are simplified to a set of literals through reduction parsing (Penberthy 1993). This approach simplifies the matching of action effects to goals. The last effect conjunct of **Fast-Fly**,

$$\forall_{\mathbf{time}}\, t\ t \in [t_s, t_e] \supset \frac{\partial}{\partial t}\mathbf{fuel}(t, \mathbf{plane}) = \nu_4$$

states that the **fuel** level will change at a constant rate of ν_4. The value of ν_4 is constrained with additional equations, relating ν_4 to the speed of the plane, 600 miles per hour, and the fuel efficiency, $\mathbf{mpg}(\mathbf{plane})$. We must specify the entire continuous behavior over the interval $[t_s, t_e]$, as our semantics insist that all continuous behaviors are the result of direct, explicit action. After an action effect terminates, the last value obtained by continous change will persist through time until explicitly modified by another action's effect.

Plans

ZENO *plans* are triples $\prec\mathcal{S}, \mathcal{L}, \mathcal{C}\succ$ where $\mathcal{S}$ is a set of steps (*i.e.*, instantiated action schemata), $\mathcal{L}$ is a set of causal links, and $\mathcal{C}$ is a set of constraints. The constraints in $\mathcal{C}$ include metric equations, linear equalities, linear inequalities, and noncodesignation constraints. The steps in $\mathcal{S}$ are partially ordered by the relevant temporal constraints in $\mathcal{C}$. The causal links $\mathcal{L}$ denote protection ranges for literals; each link is a pair $\prec\iota, \theta\succ$ where θ is a literal that must remain true throughout the interval of time ι.

Because preconditions and effects have explicit temporal scope, a planning problem can be encoded as a partial plan with a *single* dummy step whose time of "execution" bounds all planned activity. For example, our sample problem becomes the dummy step:

Schema Dummy
at-time: $[t_0, t_1]$
precondition:
$\mathbf{at}(t_1, \mathbf{scott}, \mathbf{city\text{-}d}) \wedge \mathbf{at}(t_1, \mathbf{ernie}, \mathbf{city\text{-}d})$
constraints:
$t_0 < t_1 \leq t_0 + 5.5$
effect:
$\mathbf{at}(t_0, \mathbf{scott}, \mathbf{city\text{-}a}) \wedge \mathbf{at}(t_0, \mathbf{ernie}, \mathbf{city\text{-}c}) \wedge$
$\mathbf{at}(t_0, \mathbf{dan}, \mathbf{city\text{-}c}) \wedge \mathbf{fuel}(t_0, \mathbf{plane})=500$

This unintuitive encoding of planning problems was chosen because ZENO's temporal model eliminated the need for separate initial and goal steps. When we introduce continuous time into a planning system, initial conditions, external events and domain axioms become formally equivalent. They are simply clauses that occur, beyond the program's control, at specific times. Final goals and deadline goals are also indistinguishable. They are merely clauses that must be achieved at a specific time. We lump external events, initial conditions, and domain axioms into the effects of the dummy action. Deadline goals and final goals are lumped into

its preconditions. Finally, the time of the dummy action spans the desired time for the plan to complete.

In addition, conditional effects represent external events that can be disabled, *i.e.*, one can specify that unless a bomb is disarmed by a specific time, it will explode. Domain axioms are encoded as universally quantifed temporal effects.

The Zeno Algorithm

ZENO is a least commitment, regression planner. It searches a space whose nodes are pairs $\prec P, G \succ$ where P is a partially specified plan and G is a goal agenda. As ZENO traverses arcs, it rewrites complex goals into simpler ones, satisfies simple goals, imposes constraints, and generates subgoals.

The planner begins at a node where $P = \prec \mathcal{S}, \mathcal{L}, \mathcal{C} \succ$ is a one-step plan encoding the planning problem and G is the agenda of top-level goals. The algorithm terminates when it finds a node whose agenda is empty (signifying a solution) or when the plan's constraints are inconsistent (failure). The search process is described as a flow chart in figure 3.

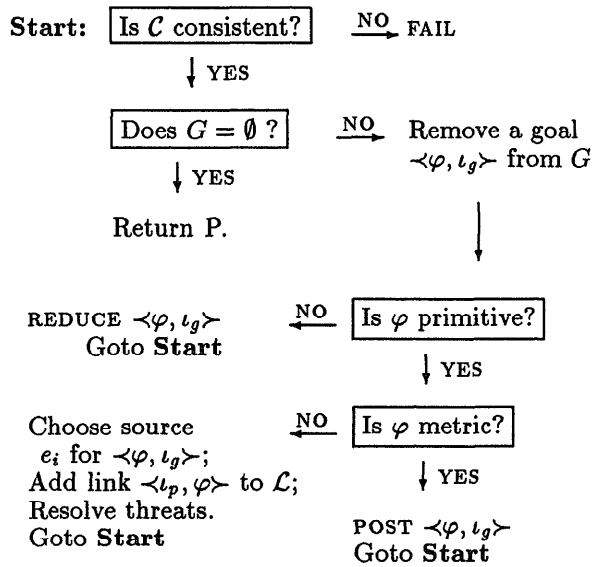


Figure 3: The main loop of ZENO.

The full algorithm involves three nondeterministic decisions: (1) decomposing a complex goal into simpler formula, (2) choosing actions to satisfy simple goals, and (3) introducing constraints to prevent interference between actions and goals. Completeness requires backtracking on these decisions — the branching factor is proportional to the number of available actions and the number of disjunctive goals. Note that completeness does *not* require backtracking on goal selection (seen as "Remove a goal" in figure 3). Since subgoal ordering decisions *can* affect planning performance as much as the true nondeterministic choices, domain dependent guidance (when available) is useful for all four types of decisions.

The remaining subsections briefly describe each path through the main loop. These paths are dispatched by first testing to see if a goal $\prec \varphi, \iota_g \succ$ remains on the agenda; if so, it is removed. Note that this tuple format is representative of the sentence $\forall_{\texttt{time}} t \; t \in \iota_g \supset \varphi$. ZENO next checks whether φ is primitive, *i.e.*, if it is a logical literal (*e.g.*, $R(x_1 \ldots x_n)$ or $\neg R(x_1 \ldots x_n)$), a metric equality (possibly constraining a fluent, *e.g.* $f(x_1 \ldots x_n) = \nu$), an arbitrary metric constraint between metric primitives (*e.g.*, $\nu_1 \leq \nu_2$) or codesignation constraint $x \approx y$. Unless φ is primitive, it is reduced as explained in the next section.

Goal reduction

The `reduce` procedure simplifies a complex goal by substituting stronger yet simpler conditions. A disjunctive goal $\varphi_1 \vee \ldots \vee \varphi_n$ is replaced, nondeterministically, by one of its disjuncts φ_i. A conjunctive goal $\varphi_1 \wedge \ldots \wedge \varphi_n$ is replaced by the set of goals $\{\varphi_1, \ldots, \varphi_n\}$.

An interval goal $\forall_{\texttt{time}} t \; t \in \iota \supset \varphi$ is reduced in one of two ways: either the program splits the interval ι into two subinterval subgoals, or it marks the interval as indivisible. This allows ZENO to explore all possible subdivisions of interval goals. Each marked interval corresponds to a linear segment of a piece-wise linear equation φ, or it corresponds to a single interpretation of logical literal φ. To avoid infinite branching, the implementation will only split intervals to a preset depth. This bound restricts the number of actions that, in combination, can be used to satisfy an interval goal; iterative deepening search can ensure completeness.

A universally quantified goal $\forall_{\texttt{type}} x \varphi$, where $\texttt{type} \neq \texttt{time}$, is replaced by its *universal base* (Penberthy & Weld 1992; Weld 1994), which is the conjunction of all ground terms φ_i, one for each extension x_i of x where x_i is a constant of type `type`. Note that domains must be finite for this to work; hence we treat time specially as described in the previous paragraph and the next section.

Existential quantifiers within the scope of a universal quantifier are replaced with Skolem functions. All other existentials are treated as simple variable names, requiring algorithms to handle codesignation and non-codesignation constraints.

Finally, metric constraints on logical fluents, such as $\texttt{value}(\texttt{t}_1, \texttt{x}) \leq \texttt{value}(\texttt{t}_2, \texttt{y})$ are separated into their individual components,[1] yielding *e.g.*

$$\exists \nu_1, \nu_2 \; \texttt{value}(\texttt{t}_1, \texttt{x}) = \nu_1 \wedge \texttt{value}(\texttt{t}_2, \texttt{y}) = \nu_2 \wedge \nu_1 \leq \nu_2$$

[1] Although this last transformation may seem trivial, subtle arguments (Nelson & Oppen 1979; Penberthy 1993) show that it is necessary to ensure soundness.

Metric and codesignation goals

If φ is a codesignation (*e.g.*, $x \approx y$ or $x \not\approx y$) or primitive metric constraint (*e.g.*, $\nu_1 \leq \nu_2$), it is posted directly to the constraint reasoning system which determines whether its constraints are collectively consistent. The phrase "POST $\prec\varphi, \iota_g\succ$" of figure 3 means the following. If ι_g is a time point, then only one constraint φ is posted, $\varphi(\iota_g)$. Otherwise, ZENO exploits piecewise linearity and posts φ for both endpoints of the interval.[2] For example, the requirement that the plane have fuel ≥ 0 during flight yields constraints that the plane have gas at takeoff and landing. If the constraint is valid at both endpoints, the Mean Value Theorem guarantees that it will be true for the entire interval.

This approach works because we limit goals φ to linear inequalities and assume that no further decomposition of ι_g is needed. ZENO handles the case where ι_g needs to be divided into subintervals in the call to REDUCE, the "goal reduction" path.

Logical and fluent-definitional goals

If φ from $\prec\varphi, \iota_g\succ$ is a literal such as `At(ernie, city-d)`, it is satisfied in a style similar to UCPOP (Penberthy & Weld 1992; Weld 1994) and SNLP (McAllester & Rosenblitt 1991) in the "logical goals" path of the main loop. ZENO nondeterministically chooses a *source* for φ by finding an effect that concludes φ over ι_e, where ι_e possibly precedes ι_g. Sources from both newly instantiated and existing steps S_i are considered. In both cases, ordering constraints are added to $\mathcal{C}$, ensuring that ι_e precedes[3] ι_g in any final plan.

ZENO then protects the literal φ over the interval ι_p, where ι_p exactly covers both ι_g and ι_e. This is accomplished by first adding a new causal link $\prec\iota_p, \varphi\succ$ to $\mathcal{L}$ and then removing all *threats* to the new link. The tuple format of a causal link is shorthand for a logical sentence stating that φ must persist over the interval ι_p (Penberthy 1993).

A threat is any effect e_k that might possibly cause $\neg\varphi$ over some portion of the interval ι_p. We say "possibly" here since the plan P is only partially specified: many step orderings and values for free variables may be consistent with P, yet allow threats to occur. ZENO resolves all threats using the standard techniques of *promotion* and *demotion*, *i.e.*, posting ordering constraints on time points (Chapman 1987), and *confrontation*, *i.e.*, posting a new subgoal that prevents e_k from interfering (Collins & Pryor 1992; Penberthy & Weld 1992; Weld 1994). If no resolution is possible, ZENO backtracks.

Linking and threat prevention introduce constraints on the plan. For example, when achieving a goal of the form `fuel(t,x)`$= \nu_g(t)$ with an effect `fuel(t,y)`$= \nu_e(t)$, ZENO must ensure that $x \approx y$ and that $\nu_g(t) = \nu_e(t)$. This ensures that any interpretation for the variables x and y are consistent with the effect "achieving" the goal. It also connects the precondition constraints on $\nu_g(t)$ to the effect constraints on $\nu_e(t)$. Similarly, if `fuel(t,x)`$= \nu_g(t)$ were defined by an effect that specified the derivative of `fuel()` over $[t_0, t_1]$ to be δ, ZENO must also constrain $\nu_g(t)$ accordingly, *e.g.*, by defining $\nu_g(t) = \nu_g(t_0) + \delta * (t - t_0)$ and posting $t_0 \leq t \leq t_1$.

Integrated Constraint Management

Since ZENO relies on constraint satisfaction for all temporal and metric aspects reasoning, sound and efficient algorithms are essential. Specialized routines cooperate to handle the different types of constraints in $\mathcal{C}$: codesignations, linear equalities, linear inequalities, and nonlinear equations.

Codesignations are handled as they were in UCPOP (Weld 1994). A simple algorithm maintains equivalence classes of all logical variables, then determines whether the noncodesignations are inconsistent with the classification.

Mathematical formulae posted by ZENO are parsed dynamically into a set of linear equations $\sum_i a_i x_i = b$, inequalities $\sum_i a_i x_i \leq b$, and pairwise nonlinear equations $x_i y_i = c$. These canonical forms are identical to the matrix representation of equations used in linear algebra and operations research (Karloff 1991).

Linear equations are solved by Gaussian elimination, linear inequalities by the Simplex algorithm, and nonlinear equations are delayed until they become linear via the solution of other equations and inequalities. To ensure sound constraint handling, each equality, $x_i = c$, that is derived by one algorithm is passed to all other algorithms (Nelson & Oppen 1979).

Determining an inconsistency using Gaussian elimination is straightforward; if a constraint $c = 0$ is detected during elimination, where c is non-zero constant, then the equations are inconsistent. Finding inconsistencies in linear inequalities is a bit trickier.

Recall that linear programming is the task of minimizing a cost function while satisfying a set of linear inequalities (Karloff 1991). The Simplex algorithm operates in two phases. First, it constructs a polytope, *i.e.*, a convex region in $\Re^n$, that exactly covers the set of solutions to the linear inequalities. In the second phase, it walks along vertices of the polytope in search of values that minimize the cost function. For ZENO, the optimization aspect is irrelevant. Instead, ZENO uses the first phase to determine simply whether the polytope is malformed. If the polytope vanishes to the null vector $\vec{0}$, no solutions exist and the constraints are inconsistent. While exponential in the worst case, the expected time for phase I is linear in the number of variables.[4] For maximum speed, ZENO uses Jaf-

[2] It is an error for ι_g to be anything but a closed interval of time $[t_1, t_2]$ or a time point t_1, by the definition of goals.

[3] More exactly, ι_e must begin before or coincident with the start of ι_g. Although ι_e may overlap ι_g, this is not required, since ZENO's threat resolution mechanism ensures that φ persists if there is a gap between ι_e and ι_g.

[4] In our experience, the Simplex algorithm is never the bottleneck; if larger problems cause this to be the case we

far *et.al.*'s (Jaffar *et al.* 1992) dynamic programming version of the algorithm optimized for incremental updates. This version retains the polytope from n equations, then modifies it when the $n+1$st inequality is added.

The above algorithms determine whether the set of constraints are consistent. However, they are not amenable to the numerous temporal queries required by the ZENO algorithm. When linking effects to goals or checking for threats, ZENO must determine whether two or more intervals overlap. These intervals, in turn, are specified as constraints on two end points. For example, the half-open interval $[t_1, t_2)$ represents all time points t such that $t_1 \leq t < t_2$. To expedite temporal queries, ZENO caches temporal relations with Warshall's transitive closure algorithm (Warshall 1962). For each time point t, this cache specifies all time points $t_{\leq}$ less than or equal to t, all time points $t_{\geq}$ greater than or equal to t, and all time points $t_{\neq}$ distinct from t. This can be efficiently implemented using boolean operations on bit vectors, where each time point is represented by a unique index.

Formal Properties

Assuming that all interval, metric effects are piecewise linear, that nontemporal types are static and finite, and that nonlinear equations can be linearized, then ZENO is both *sound i.e.*, all plans returned by ZENO will work, and *complete i.e.*, if a plan exists, ZENO will find it. The soundness proof introduces a loop invariant maintained by all control paths of ZENO. The halting conditions, in combination with the invariant, guarantee that every plan returned by ZENO will work. The completeness proof uses induction on the number of steps in a plan. The base case (0-step plans) is true for all consistent problem descriptions. The inductive case uses an $n-1$ step plan to guide ZENO as it builds an n step plan.[5] ZENO's proofs occupy many more pages than allowed in this paper; see (Penberthy 1993) for details.

Performance

ZENO has been tested on numerous problems. Our empirical results (Penberthy 1993) show that ZENO's performance is on a par with state-based planners, *e.g.*, UCPOP and PRODIGY, in domains that don't involve interval goals, continuous change and metric relationships (which those planners can't handle). ZENO's speed only degrades when a planning problem demands ZENO's advanced features. Yet even in these domains performance is tolerable, *i.e.*, the current implementation is suitable for experimental research use. Further work on search control and abstraction is needed before ZENO can handle large-scale, practical problems.

could switch to Karmarkar's linear programming algorithm which is guaranteed polynomial (Karloff 1991).

[5]This *clairvoyant* proof technique was first used by McDermott (McDermott 1991) to prove his total-order planner complete.

Figure 4 shows how ZENO performs[6] on three such problems: Allen's door latch example (Allen *et al.* 1991), the airplane example of this paper and the metric blocks world problem from figure 9.3 of (Wilkins 1988b). A simple predicate ordering, *e.g.*, see (Sacerdoti 1974), was used on all but the door latch problem to guide subgoal selection. Iterative-deepening, depth-first search (Korf 1985) handled all other nondeterministic choices.

PROBLEM	CPU TIME (SEC)
Door latch	0.04 ± 0.00
Airplane routing	151.56 ± 27.56
SIPE example	1.42 ± 0.24

Figure 4: Execution times for the ZENO planner.

Related Work

Because of our interest in formal properties, ZENO is closest in spirit to the work of Allen(Allen 1991), Chapman(Chapman 1987), McAllester(McAllester & Rosenblitt 1991) and Pednault(Pednault 1986). Allen and Pelavin (Allen *et al.* 1991) describe an elegant theory of temporal planning based on first order logic and an interval model of time. In contrast, we model time using the real numbers; this allows metric duration and continuous change.

Numerous systems with some of ZENO's features have been implemented in the past twenty years and we have drawn insight from many of them. Drabble's EXCALIBUR (Drabble 1993) first generates a plan that ignores metric constraints, then tests it through qualitiative simulation; failed tests invoke heuristic replanning. Simmons' GORDIUS (Simmons 1988) handles actions with conditional and metric effects, but uses a state-based model of time and is incomplete. Our approach is considerably simpler than that of SIPE (Wilkins 1990) and DEVISER (Vere 1983) – ZENO avoids parallel links, complex traversal schemes, and heuristic plan evaluation. Similarly, we believe that ZENO's treatment of simultaneous and metric effects is more general than SIPE's. While OPLAN (Currie & Tate 1991) uses ideas from operations research to optimize resource usage, they use different techniques for temporal management. In contrast, ZENO uses an integrated approach for both temporal and other metric constraints, but makes no claim of efficient resource handling.

(Jaffar *et al.* 1992) developed the incremental algorithms and the idea of using Gaussian elimination and Simplex phase I iteration to manage linear equations and inequalities. Our restrictions on the use of metric variables in ZENO's logic are derived from the innovative approaches of (Nelson & Oppen 1979) and (Hendrix 1973).

[6]The experiments were performed on an IBM RS/6000 running Allegro Common Lisp; 95% confidence intervals were calcuated from 10 runs per problem.

Conclusion

ZENO is a least commitment, refinement planning algorithm capable of handling simultaneous actions, continuous change, metric reasoning and deadline goals. Both actions and goals are described in a rich logic supporting universal quantification, disjunction, conjunction, metric functions, logic functions and formal objects. The algorithm is sound *i.e.*, all plans returned as solutions are guaranteed to work. The algorithm is also complete *i.e.*, if a plan exists, then ZENO will find it. A full, working implementation of the program has performance similar to existing state-based planners on comparable domains, but cannot be said to have *heuristic adequacy.* We strive for a system with the performance of SIPE (Wilkins 1988a) and the formal properties of ZENO. Since metering tools show that the bulk of ZENO's time is spent updating and querying its temporal cache, we hope to integrate optimized temporal reasoners, such as (Dechter, Meiri, & Pearl 1991),(Dean 1989) or (Williamson & Hanks 1993), into ZENO's hierarchy of constraint reasoners. As it stands, we believe that ZENO represents a first step towards bridging the gap between formal and empirical approaches to automated planning with expressive temporal languages.

Acknowledgments

We thank Tony Barrett, Alan Borning, Ernie Davis, Denise Draper, Oren Etzioni, Keith Golden, Steve Hanks, Nick Kushmerick, Edwin Pednault, Ying Sun, Mike Williamson, and the anonymous reviewers for helpful comments.

References

Allen, J., Kautz, H., Pelavin, R., and Tenenberg, J. 1991. *Reasoning about Plans.* San Mateo, CA: Morgan Kaufmann.

Allen, J. 1991. Planning as temporal reasoning. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, 3–14.

Chapman, D. 1987. Planning for conjunctive goals. *Artificial Intelligence* 32(3):333–377.

Collins, G., and Pryor, L. 1992. Achieving the functionality of filter conditions in a partial order planner. In *Proc. 10th Nat. Conf. on A.I.*

Currie, K., and Tate, A. 1991. O-plan: the open planning architecture. *Artificial Intelligence* 52(1):49–86.

Dean, T. 1989. Using Temporal Hierarchies to Efficiently Maintain Large Temporal Databases. *Journal of the ACM* 36(4):687–718.

Dechter, R., Meiri, I., and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–96.

Drabble, B. 1993. Excalibur: a program for planning and reasoning with processes. *Artificial Intelligence* 62:1–40.

Hendrix, G. 1973. Modeling simultaneous actions an continuous processes. *Artificial Intelligence* 4:145–180.

Jaffar, J., Michaylov, S., Stuckey, P., and Yap, R. 1992. The CLP($\mathcal{R}$) Language and System. *ACM Transactions on Programming Languages and Systems* 14(3):339–395.

Karloff, H. 1991. *Linear Programming.* Boston: Birkhäuser.

Korf, R. 1985. Depth-first iterative deepening: An optimal admissible tree search. *Artificial Intelligence* 27(1):97–109.

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proc. 9th Nat. Conf. on A.I.*, 634–639.

McDermott, D. 1991. Regression planning. *International Journal of Intelligent Systems* 6:357–416.

Nelson, G., and Oppen, D. C. 1979. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems* 1(2):245–257.

Pednault, E. 1986. *Toward a Mathematical Theory of Plan Synthesis.* Ph.D. Dissertation, Stanford University.

Penberthy, J., and Weld, D. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, 103–114. Available via FTP from `pub/ai/` at `cs.washington.edu`.

Penberthy, J. 1993. *Planning with Continuous Change.* Ph.D. Dissertation, University of Washington. Available as UW CSE Tech Report 93-12-01.

Sacerdoti, E. 1974. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence* 5:115–135.

Simmons, R. 1988. Combining associational and causal reasoning to solve interpretation and planning problems. AI-TR-1048, MIT AI Lab.

Vere, S. 1983. Planning in time: Windows and durations for activities and goals. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 5:246–267.

Warshall, S. 1962. A theorem on boolean matrices. *Journal of the ACM* 9(1).

Weld, D. 1994. An introduction to least-commitment planning. *AI Magazine.* Available via FTP from `pub/ai/` at `cs.washington.edu`.

Wilkins, D. 1988a. Causal reasoning in planning. *Computational Intelligence* 4(4):373–380.

Wilkins, D. E. 1988b. *Practical Planning.* San Mateo, CA: Morgan Kaufmann.

Wilkins, D. 1990. Can AI planners solve practical problems? *Computational Intelligence* 6(4):232–246.

Williamson, M., and Hanks, S. 1993. Exploiting domain structure to achieve efficient temporal reasoning. In *Proc. 13th Int. Joint Conf. on A.I.*, 152–157.