July 1988

Constraint-Based Temporal Planning

Norman I. Badler
University of Pennsylvania, badler@seas.upenn.edu

S. Kushnier
University of Pennsylvania

Jugal Kalita
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-88-55.

Constraint-Based Temporal Planning

Abstract

This paper deals with the application of notions from "planning" and the "representation of temporal information" in an animation system to simulate human task performance. Specifically, a model was developed in which the representation and manipulation of temporal information forms the basis of a planning system. This model has been implemented as part of an animation system in which the goal was to enable a natural, clear language for the specification of relevant temporal constraints along with a planner that would manipulate these constraints, ultimately resulting in exact timing parameters to be passed to the low-level animation routines. The first part of the paper describes the essential role of temporal planning in an animation system that models human task performance. The second part of the paper goes on to explain how a wide variety of temporal constraints can be "compiled" into a set of low-level "simple constraints" through the use of "dummy intervals" and "fuzzy constraints." The third part of the paper describes how a definite "plan" of events can be generated based on an analogical "spring system."

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-88-55.

CONSTRAINT-BASED TEMPORAL PLANNING

N. Badler, S. Kushnier and J. Kalita

MS-CIS-88-55
GRAPHICS LAB 24

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104

July 1988

Acknowledgements: This research is partially supported by Lockheed Engineering and Management Services, the Pennsylvania Benjamin Franklin NASA Grant NAG-2-426 Partnership, NSF CER Grant MCS-82-19196, NSF Grants IST-86-12984 and DMC-85-16114, and ARO Grants DAA29-84-9-0027, DAAG29-84-K-0061 including participation by the U.S. Army Human Engineering Laboratory.

Constraint-Based Temporal Planning

N. Badler, S. Kushnier, and J. Kalita
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104-6389

July 20, 1988

Abstract

This paper deals with the application of notions from "planning" and the "representation of temporal information" in an animation system to simulate human task performance. Specifically, a model was developed in which the representation and manipulation of temporal information forms the basis of a planning system. This model has been implemented as part of an animation system in which the goal was to enable a natural, clear language for the specification of relevant temporal constraints along with a planner that would manipulate these constraints, ultimately resulting in exact timing parameters to be passed to the low-level animation routines. The first part of the paper describes the essential role of temporal planning in an animation system that models human task performance. The second part of the paper goes on to explain how a wide variety of temporal constraints can be "compiled" into a set of low-level "simple constraints" through the use of "dummy intervals" and "fuzzy constraints." The third part of the paper describes how a definite "plan" of events can be generated based on an analogical "spring system."

1 Introduction

While *planning* and *representation of temporal information* have been studied as independent fields for the most part, there are many problems central to both fields. In addition, the weaknesses of many planning paradigms seem to lie in the lack of a good model of time. Similarly, many representation models include very little concerning the actual application of this knowledge (as in some kind of planner). In particular, most planning systems include a great deal of *processes*, capable of handling a large class of planning problems; however, the representations used tend to be "ad hoc" and there is little notion of the importance of *time* itself, upon which the planner rests. On the other hand, most schemes for the representation of temporal information treat temporal structure explicitly, while being unable to actually *do* anything with that knowledge other than manipulate it in various ways. Our particular approach is to deal explicitly with aspects of both planning and time representation in a unified framework.

In general, there is little uniformity in the treatment of temporal information. Most work seems to be "ad hoc" and tailored specifically to the task at hand. For example, in Vere's planning system [9], each planned event has associated with it various constraints concerning boundaries on starts and ends, while the durations are specified as constants. Then, a *constraint propagation* algorithm is used to find a consistent plan of events. In this model the temporal aspects are simply modelled "on top of" an already existing planner. This approach is limited because any model built up in such a way is constrained by a whole class of assumptions implicit in how the underlying planner deals with time.

A more formal approach to the manipulation of temporal information is McDermott's "situation calculus" [5, 8]. In models of this kind, all temporal information is represented in the form of logical propositions, while the manipulation consists of inference rules that state how new information can be deduced from the existing information. The main problem with these approaches is that the information and inferences are inextricably linked: the propositions are given meaning only by the inferences that act on them and the inferences are applicable only to information in a particular form. This is in contrast to theories that state an explicit representation, upon which various processes may operate, some more efficient or elegant than others.

Allen has proposed such a model. In Allen's system [1], the basic unit is the "time-interval". Temporal information is represented by collections of possible relations that may hold between these *intervals*. In this model, a transitivity table specifies how relations may be maintained through the propagation of individual relations. For example, once it is

known that interval A is *before* interval B, and that interval B is *before* interval C, it can be deduced that interval A is *before* interval C. Allen goes to great lengths to show that his basic representation is intuitively sound, formally sound, and representative of real-world temporal phenomena in problem-solving and planning environments [2, 3]. However, it is not clear how this model can be applied so as to result in an explicit "plan" of events. One problem concerns the fact that disjunction is simply represented as a collection of possibilities, while a *planner* must ultimately "choose". Another problem is that Allen's model is purely relational and does not include representation of durations and *real times*, such as "1:00PM" and "ten minutes".

Malik and Binford [7] have proposed a model in which they show that all temporal information can be expressed in terms of constraints on endpoints, and point out that *linear programming* can then be used to "solve" such systems. Their approach is notable in that it can be solved computationally, rather than deductively with an inference engine or through symbolic manipulation. However, their approach does not allow for a measure of uncertainty in specification. Each constraint is all-or-nothing and the final plan must satisfy each constraint exactly, resulting in a system which is relatively inflexible. With such a model, it is often difficult to specify *enough* information without overconstraining the system by specifying *too much*.

In this paper, we present a scheme for representation of time as an integral part of event and process description, keeping in mind that our proposed method must be amenable to expedient handling by a planning system. Following Allen [1], intervals are taken as the fundamental representation units. However, the novelty of this approach lies in the manner in which interval information is manipulated to satisfy flexible planning of events and processes. Relationships are represented numerically and a *quadratic programming* solution is used to obtain temporal variables relevant to the low-level animation routines. While *representation* schemes such as Allen's seem to be too flexible to result in any kind of useful plan and *planning* schemes seem too inflexible to allow for a natural specification of information, our system is a compromise, allowing for *real* time specifications and flexible planning through a computational algorithm.

2 Overview

While an animation, in principle, can be approached strictly from the level of positioning and motion of human figures in space, this level of specification results in an animation

sequence that is relatively inflexible. It is extremely difficult for the general user to specify in precise detail how the animation is to progress. In addition, modification of high-level aspects of the animation often entails the specification of an entirely new animation sequence. What is needed is a sufficiently high level of description such that the "animation details" are taken care of in the animation itself. In short, the specification should be at the level of *what* is to be animated, as opposed to *how* it is to be animated [6]. Such a level of description lends itself nicely to notions from the study of *artificial intelligence*. Such tools as a powerful knowledge base and a natural language parser [6] do much to bridge the gap between high-level specification and low-level graphical processes, resulting in an improved level of description and degree of flexibility. In the same spirit, this work allows for a level of description of temporal information at a level applicable to events and processes. A planner can then used to compute the details of exactly when each event must take place.

There are some other good reasons to incorporate notions from artificial intelligence in a human task simulation system. To begin with, in order that the animation look natural and realistic, it is not enough to simply animate human figures; one must animation *intelligent* human figures. Animated figures that do not perform tasks as humans do, respond to "stimulus", or cooperate, will appear robotic and lifeless. Hopefully, the incorporation of some kind of "intelligence" pertaining to task performance will result in the *appearance* of intelligence in the final animated product. Furthermore, animation could prove to be an ideal "testing ground" for advances in artificial intelligence. This type of simulation provides for a "world" that is sufficiently *real* that the ultimate output is much more digestible than text output or "traces". Meanwhile, the "world" is sufficiently *unreal* that real-life problems such as *real-time* issues and *noise* do not require handling to the extent that they do in real-world robotics or perception.

This paper tackles notions from *planning* and *temporal representation*, as they might be applied in an animation system. Issues of *when* events are to be animated take many different forms and are essential to the animation as a whole. First of all, there are many domains in which explicit temporal constraints are required and must be incorporated into the animation accordingly. For example, a space shuttle mission often includes a complex chronology of specific events. Also, if cooking, a recipe often calls for specific temporal constraints such as how long to bake, etc. In order to animate such action sequences, the relevant temporal information must be *specifiable* and brought to bear in the *planning* of the animation.

In addition, the modelling of complex human tasks often includes a decomposition of events, all of which are temporally related in various ways. For example, in unlocking a

door, one must first get the key, then insert the key, then turn the key. Each sub-event must follow "on the heels" of the preceding sub-event. Orderings such as turning the key before inserting the key or inserting the key and suddenly deciding to take a walk, are bound to cause problems. The relevant temporal constraints *must* be realized in the planning system.

Furthermore, there are issues involved in how people *think* about time and specify temporal information in natural language. This is important because the specification should be at the level at which people naturally think about and specify time. This is especially important in light of the fact that much of this specification will be accomplished through a natural language interface. One specific example of where this comes into play concerns words such as "about" and "around". These *fuzzy* terms seem to defy computational modelling; however, they must mean *something* and we include a method of modelling these "uncertainty terms".

Finally, because all aspects of *when* events are to be animated are controlled by the planner, the planner must deal with notions of coordination of different processes, events that must not overlap in time (such as making breakfast and taking a shower), the synchronization of various events, and extended task descriptions. We present a model of temporal specification and planning such that all of the issues mentioned so far are addressed in a single, unified theory.

3 Specification

The basis of this model is that each event that is to be animated have associated with it a "time-interval". The *time-interval* consists of the continuous points of time over which that event is to occur. Each time interval can be described by its *start-time* and its *end-time*. The main idea is that the temporal specification include all the "relevant" constraints on when the events are to be animated. The planner's job is then to find a set of start and end times consistent with the high-level constraints. The question to be answered in this section concerns the types of temporal information required such that one can easily specify those aspects relevant to the event and process level of the animation.

3.1 Fuzziness

The simplest kind of temporal specification deals with exact start, end, and duration times. Specifications such as *"Lunch is from 12PM to 1PM"* and *"Bake for two hours"* contain vital temporal information that must find its way into the final plan. However, many times, this type of all-or-nothing constraint can cause problems both in the representation (where such *exacting* specification might not be called for), and in the solution algorithm (where there is a fine line between "underspecifying" and "overspecifying"). Our model includes a new type of constraint wherein each constraint has attached to it, a certain "constraint strength". For instance, to *bake* a cake might require a strictly timed two hours, while the time taken to *ice* the cake is less critical. In solving for such a system, compromises can often be struck such that the stronger constraints are afforded more importance than the weaker constraints. in solving for the final plan. Furthermore, a measure of the *uncertainty* or *fuzziness* in a specification can be represented as the inverse of *constraint strength*. Therefore, an uncertain temporal specification such as "around 3PM" is given a relatively weak *constraint strength*, whereas "exactly ten minutes" is given a very strong one. It is our belief that such *fuzzy* information must be represented in any complete account of temporal manipulation, and that modelling them as *constraint strengths* actually facilitates a solution by allowing for a degree of flexibility.

Another reason for modelling *uncertainty* is that people do, in fact, give temporal information with uncertainties included. One must assume that the uncertain terms such as "about" and "around" actually do *mean* something, and that they should be included in the representation *somewhere*. Even "exactly" must be regarded as a term giving uncertainty information, namely that there is none. This suggests that even when an *uncertain term* is not explicitly stated, it is almost always implied. Just as the quantity represented by "4" is somehow different from the quantity represented by "4.000", "ninety minutes" seems to imply a greater degree of precision than "an hour and a half". Furthermore, it makes little sense to try to explain this uncertainty in terms of a *tolerance*. This only postpones the problem of using an all-or-nothing constraint to model a notion which intuitively *fuzzy*. The duration represented by "about an hour" involved fuzzy shades-of-gray, rather than a clear black-and-white border.

3.2 Temporal Relations

The modelling of relations between different intervals is also an aspect of representation that needs to be dealt with. The simplest relations between two *time-intervals* are the following:

one interval immediately following another, two intervals having the same *start-time*, and two intervals having the same *end-time*. For example, in specifying the two intervals over which two people run a race, the race could start anytime and last from five minutes to five days; however, one should at least be able to specify that each racer must start at the same time. In the same light, one might wish to specify that one event must occur before or after another event, even though one has no idea how long the delay will be. For instance, *"Grasp the cup before drinking from it."* provides an explicit ordering while leaving open the question of when each will occur and what will occur between them. Therefore, one might grasp the cup and immediately drink, but one may also grasp the cup, walk into the other room, sit down, and *then* drink. These types of specifications are often useful in the temporal specification of the decomposition of tasks into subtasks, where one might not know precisely when anything is to occur, although one has a good idea of the temporal relationships between the different events.

In addition, one might wish to specify exact "bounds" on when an event is to start or end, or how long it is to last as in *"One must go there at some point between 9AM and 12AM"*, and *"Spend no less than one hour and no more than two there"*. This would be the case where the sole criteria is that certain boundaries are not crossed; within the "legal span", there are no preferences. This is not to be confused with the *fuzzy specifications*, where there are definite preferences, but no explicit boundaries. Finally, combinations of the above specifications must be supported as in *"Wait at least twenty minutes after running the dishwasher before taking a shower"*. This expresses a *relationship* between two events such that there must exist a bounded delay between them.

As a side issue, for reasons of flexibility in specification, a functional specification should be substitutable for any definite time value as used above. For instance, the time taken for agent X to run a mile might be significantly different from the time taken for agent Y to run a mile. In order that the specification of *run-a-mile* be "agent-independent", the duration must be specified functionally as follows: find the *run-rate* of the *current-agent* and divide by 1 mile. This would require functional specifications and access to a knowledge base containing information about various aspects of the "animation world".

3.3 Temporal Decomposition

As it turns out, a large variety of complex temporal constraints can be modelled by breaking them down into a series of interrelated simple temporal constraints. Intervals can be created and linked into the system of constraints in simple ways so as to maintain complex con-

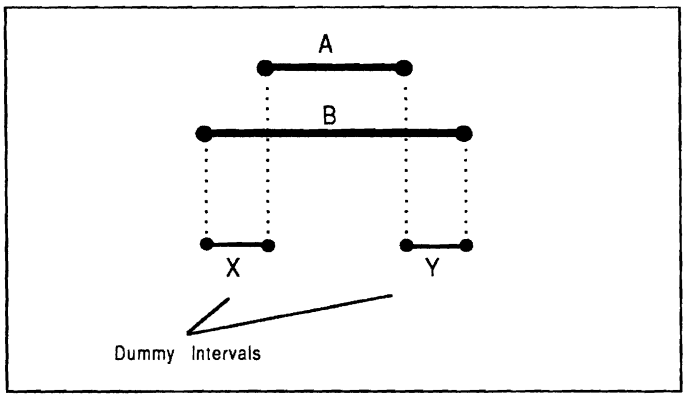

Figure 1: Function of dummy intervals

straints over other intervals; these are called "dummy intervals". For example, in Figure-1, the *dummy interval* X insures that the start of B precedes the start of A, while the *dummy interval* Y insures that the end of B follows the end of A. Note that this system can be modelled through "simple" relations between intervals; however, the strategic "simple" relations involving A, B, X, and Y guarantee that a "complex" relationship exists between A and B, namely that A occurs *during* B. In similar fashion, many other types of complex constraints can be modelled through the use of appropriately modelled *dummy intervals*. For example, by using the *duration* specification of a *dummy interval*, one can express constraints on the delays between events. For instance, in Figure-1, one can specify that the duration of X be ten minutes long, insuring that exactly ten minutes pass between the start of B and the start of A.

This breaking down of complex temporal constraints into simple temporal constraints suggests that one can specify complex temporal information such as ordering information, delays, bounds, etc., in a "high-level syntax". Then a *compiling* process can reduce this to constraints in a "low-level syntax" through proper instantiation and specification of *dummy intervals*. The result would be a "low-level syntax", easily processable while retaining all of the complex relationships specified in the "high-level syntax". In fact, the division into two levels of specification and a *compiling* process carries many advantages. A *high-level syntax* allows for intuitive, readable, free-form style representation, on the level of a task representation or natural language interface. The *low-level syntax* would be structured, contain numerical rather than symbolic values, and be amenable to a computational algorithm. The *compiling process* would take care of bounding symbolic values to numerical values, structuring the information, evaluating the *functional* specifications, and linking in the necessary *dummy intervals* on the fly.

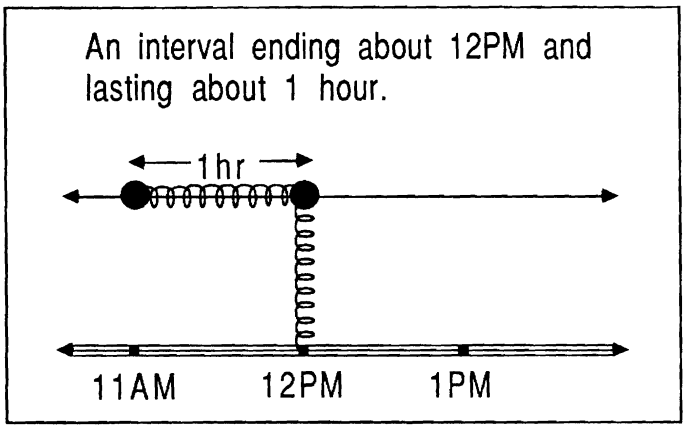

Figure 2: Simple spring system

4 Solution Method

Now that we have a *low-level representation* consisting of a set of intervals with both *fuzzy* and *exact* constraints on the starts, ends, and durations of those intervals, along with a set of equalities "tying" the starts and ends of specific intervals together, we can go about the task of finding an algorithm to solve for a *plan* in which all of these constraints are met (more or less).

4.1 Spring Analogy

Exact constraints can be modelled relatively straightforwardly as "equality constraints": $(A_{start} = 12)$. "Simple" constraints between intervals can also be modelled as such: $(B_{start} = A_{end})$. In order to understand the *fuzzy constraints*, though, the analogy of a "spring system" can be useful. The system of constraints of various strengths can be modelled as a kind of physical spring system, where each constraints is modelled as a spring that *pushes* or *pulls* the intervals around, each attempting to satisfy its constraint. For example, in Figure-2, the duration constraint acts like a spring with *natural length* of **one hour** inserted between the two endpoints, while the **12PM** end-constraint acts as if there were a vertical spring, pulling that endpoint toward **12PM**. With no other springs involved, the springs would be at rest as seen in the figure; however, were conflicting constraints introduced, they would serve to pull the endpoints this way or that. This is where flexibility and compromise enters the picture.

In the end, the amount of shrinking or stretching for each spring would be directly related to the relative *strengths* of the conflicting constraints, encoded by the *elastic spring constants*

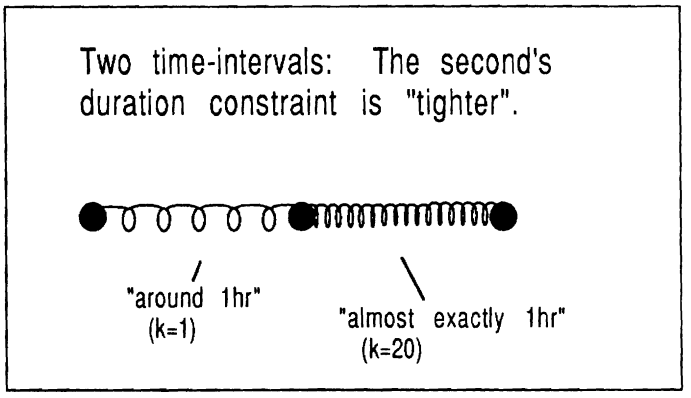

Figure 3: Analogy to Constraint Strength

of the various springs. When the constraints are all specified, what results is a system of springs, each one pushing (or pulling) to exert its own constraint, and simultaneously being pushed (or pulled) by its neighbors. Furthermore, because *constraint strength* is encoded by *stiffness*, the resulting system will be the *fairest* for each constraint. Therefore, a "general preference", such as a prefered time to wake up in the morning, would be modelled by an extremely elastic spring, quite amenable to deformation in case of conflicts. Similarly, a "critical constraint" such as the time to start work, would be modelled by a stiff spring, resistant to deformation, "pushing around" conflicting constraints so that its constraint is met.

Not only does the concept of an analogous spring system for a given system of constraints aid in clarifying the representation, but the analogy goes further. The system of springs can easily be seen to exhibit behavior that would be desirable in a planner that tries to find a set of begin and end times, given a system of varying strengths of constraints. To start with, an individual constraint should always be met if no conflicting constraints are specified. If some are, a compromise must occur in which each constraint is partially met, in accordance with its *strength.* An individual spring also meets this criteria; in isolation it will remain in its natural state; however, if forces are exerted by a neighboring spring, both will deform according to their respective elasticity constants. In addition, in a system of connected springs, a force acting on any spring may cause the whole system to deform to a new arrangement. This can be seen as a kind of *constraint propagation*, where the effects of a single constraint may be "felt" throughout the network.

Another consideration concerns a measure of *how well* a constraint is met. It is fairly obvious that a constraint that is easily met will have a corresponding spring with little energy. Again, in the larger view, the energy of a system of springs is given by the sum of the energies of each individual spring. Therefore, the best solution to a system of constraints

is met exactly when the corresponding spring system holds the least energy. This statement makes sense because, intuitively, in order to solve the constraint problems, one would want to set up the analogous spring system as specified, watch the springs "bounce around", wait until the system comes to a halt (state of minimum energy), and read off the values to get a solution. Fortunately, this fact allows for a computational solution to the constraints given: simply set up the energy equations, parameters being the start and stop values of each time interval, and minimize the equation.

4.2 Solution Method

Witkin, Fleischer, and Barr [10] also formulate a model in which a series of constraints is met through setting up energy functions and computationally minimizing the total energy. Their model is similar to ours in that they are trying to solve a system of constraints in *three-dimensional space*, while we are working in *one-dimensional time*. However, while their approach with energy constraints is used as a method of solution and an elegant way to generate an animation, we use energy constraints to explicitly model uncertainty in temporal information. Furthermore, while Witkin, et al, expect constraints to be partially met only in "overdefined" systems, we are *relying* on this occurrence.

In pure form, for a system of n time intervals, the function to be minimized is as follows:

$$E_{tot} = \sum_{i=1}^{n} \Big((k_{b_i}/2)(x_{b_i} - a_{b_i})^2 + ((k_{e_i}/2)(x_{e_i} - a_{e_i})^2 + (k_{d_i}/2)(x_{b_i} - x_{e_i} - a_{d_i})^2 \Big)$$

$\mathbf{a_b}$,$\mathbf{a_e}$,$\mathbf{a_d}$: values for start, end, and durations constraint vectors
$\mathbf{k_b}$,$\mathbf{k_e}$,$\mathbf{k_d}$: values for start, end, and durations constraint strength vectors
$\mathbf{x_b}$: resulting vector of start times
$\mathbf{x_e}$: resulting vector of end times

Note that "no constraint" on some aspect of an interval can be seen as a constraint with "very high" fuzziness where $k = 0$. Since this causes the term to drop out of the equation anyway, this need not be included in the equation. Note also that in constraints that are "nailed down", k should tend toward infinity. This is needed for such constraints as: END(A)=START(B). As this causes computational problems, this type of constraint is best modelled as an "equality constraint". Thus, we have a minimization problem in a space defined by the *equality constraints.*

Note however, that the equation says nothing about the fact that the start of any interval must precede its end. In fact, this is extremely important because all complex ordering

information rests of the integrity of the *dummy intervals*. Unfortunately, a straightforward minimum energy solution might very well include intervals that have been "flipped". The only way to deal with this problem is to include a series of "inequality constraints" of the form $(x_{b_i} < x_{e_i})$. Therefore we want to solve for the minimum of an equation of $(2n)$ parameters with possible equality and inequality constraints, both of which are linear. This happens to be a common problem in optimization theory, a *quadratic programming problem*, for which a variety of computation solution methods are known.

Our *quadratic programming* solution is basically the solution method given by Hildreth and D'Espo [4]. We have included lagrangian multipliers in the quadratic expression in order to model the *equality constraints*. Hildreth and D'Espo show how to convert the basic problem into what is called the "dual problem", which is easily solved by iterating on the lagrangian multipliers which correspond to the *inequality constraints*. The iteration can be done in such a way that the intervals that have been "flipped around" are eventually fixed, while those that are valid are simply left alone. Finally, the solved *dual problem* can be translated back to give the results of the original problem.

The first and last parts of the solution consists of translating to and from the *dual problem*. This involves nothing more complex than multiplying and inverting matrices: therefore, a problem of $O(n^3)$. The iteration involved in the *dual problem* involves a linear search, where each iteration involves an $O(n^2)$ operation. Therefore, it can be presumed that this complexity is also $O(n^3)$. Thus, the complexity of the entire solution is $O(n^3)$.

4.3 Some Examples

This example represents the planning of two time intervals A and B. The high-level syntax is self-explanatory:

```
A: ((start (around 1PM))
    (duration (about 4hr))
B: ((start ((about 3hr) before (end A))
    (end (around 7PM))
    (duration (about 3hr))
```

Converting to low-level syntax, the delay between the start of B and the end of A is expressed in the *dummy interval* X.

```
A: ((1 1) nil (5 4))
B: (nil (1 7) (5 3))
```

```
X: ((B start) (A end) (5 3))
```

The respective energies of each spring:

$$E_1 = (1/2)(x_{b_1} - 1)^2 + (5/2)(x_{e_1} - x_{b_1} - 4)^2$$

$$E_2 = (1/2)(x_{e_2} - 7)^2 + (5/2)(x_{e_2} - x_{b_2} - 3)^2$$

$$E_3 = (5/2)(x_{e_3} - x_{b_3} - 3)^2$$

Equalities: $x_{b_3} = x_{b_2}$, and $x_{e_3} = x_{e_1}$.

Inequalities: $(x_{b_1} < x_{e_1})$, $(x_{b_2} < x_{e_2})$, and $(x_{b_3} < x_{e_3})$

Solving results in the following:

```
A: (1:46PM 5:55PM)
B: (3:05PM 6:14PM)
X: (3:05PM 5:55PM)
```

Note that since all the constraints could not be met simultaneously, none of the constraints were met exactly. However, the solution is surprisingly good. The duration constraints are all deviated by nine or ten minutes, while the preferred 1PM and 7PM are deviated by 46 minutes.

The following example would model a person, who would prefer to get up at 10AM, start work at 9AM, and work for three hours. Unfortunately, other constraints conflict. Also, time must be allotted for getting ready for work.

```
Wake-up:  ((start (after 6AM))
                  (pref 10AM))
           (duration (exactly 6min))
Prepare:  ((start (after (end Wake-up)))
           (duration (around 1hr))
Work:     ((start (before 9AM) (pref 9AM))
           (end (after 5PM))
           (duration (pref 3hr)))
```

Solving results in the following:

```
Wake-up:  (8:05AM  8:11AM)
Prepare:  (8:11AM  9:00AM)
Work:     (9:00AM  5:00AM)
```

One can see that the person's preferences in work duration are irrelevant; however, his

preference to sleep late causes him to *wake-up* later than otherwise, causing him to rush through *Prepare*, in order to get to work on time.

4.4 Relation to Other Models

Relation to Allen's work: Allen has gone to great lengths to show that his basic representation is intuitively sound, formally sound, and applicable to real-world temporal phenomena [2, 3]. Although we build on this representation in a different way, our representation is consistent with Allen's work at a formal level.

As far as relations between intervals, while Allen takes the thirteen possible interval relations as his basic level of representation, our representation represents relations directly by constraints on the endpoints. This is consistent with some of Allen's work on the relations between *time intervals* and *time points* [3]. For example, to model a MEET relation between A and B, we specify that END(A)=START(B). In addition, some relations are represented more naturally in our representation. While Allen must represent the fact that two intervals start at the same time by a disjunction of three relations (representing the possible differences in ordering of end times), our representation can model this directly by specifying the start times equal: START(A)=START(B).

Another way in which our representation is consistent with Allen's work concerns the concept of the *dummy interval*. Allen has shown that all of his interval relations can be built up from the one relation 'MEET' [3]. For example, the 'BEFORE' relation can be defined as follows: $(A < B) \equiv \exists X[(A\ m\ X) \wedge (X\ m\ B)]$. We turn this around and *insure* that $(A < B)$ by explicitly *creating* the interval X, linked to A and B appropriately: START(X)=END(A) and END(X)=START(B).

One notable difference is that while our system is designed with *real* time values in mind, Allen's representation is purely relational and flexible in a topologic sense. This seems reasonable in light of the fact that any references to "real times" might be seen as automatically invoking a *moment-based* representation scheme, complete with all its logical inconsistencies. However, our theory claims to be basically interval-based, while including references to *real times*. Our goal is to keep the representation as clean as possible, while still being able to utilize the underlying mathematical structure of the real numbers to generate a solution.

Relation to Vere's work: Many relations similar to Vere's *window* constraints [9] can be expressed in our framework. We can easily represent boundaries on start and end

times, as well as exact duration constraints. Constraints that specify *optimal* values can be expressed as well, through using "very weak" constraint strengths. However, where Vere's *optimal* constraints are met if possible and otherwise ignored, our constraints "try" to be satisfied, even if they can not possibly be met.

Relation to Malik and Binford's work: While Malik and Binford [7] mention *linguistic fuzziness* with respect to temporal specification, they do not incorporate it into their model. Although (as they express) such phrases as "a while ago" may be too vague to express, it seems clear that a phrase such as "about ten minutes" carries *real* information, along with a degree of uncertainty. Our work can be seen as a generalization of theirs, where we allow for flexible rather than exact constraints, incurring a problem which requires a *quadratic programming* solution, rather than a *linear programming* solution.

5 Discussion

5.1 Evaluation

One of the main ideas to come out of this work is the idea that complex temporal information can be decomposed into simple temporal information through the use of *dummy intervals*. Not only does this provide for two separate levels of specification, a high-level user-oriented specification and a low-level system-oriented specification, but this also introduces a compiling process, into which one can specify how symbols are to be represented numerically and how complex information is to be broken down. This provides for an additional level of flexibility, allowing for modifications as to the high-level symbolic representation without needing to modify the essential planning algorithm.

Anther useful idea is the modelling of *fuzzy* specifications through the spring analogy. This provides a model which is intuitively satisfying, while allowing for flexible planning and ultimately resulting in an algorithm for generating exact start and end times for events. It also represents a new approach to constraint specification, such that constraints are not all-or-nothing, but result in a graded structure of better and worse constraint satisfaction.

Finally, a distinguishing mark of this work is the fact that the planning algorithm is numerical. This carries many advantages, but also some disadvantages. The first advantage is in terms of efficiency; numerical algorithms are generally much faster than the symbolic manipulation or inference engines currently used in much of AI. This is especially important

as this system is to be used inside an animation system, where formal correctness may be sacrificed for speed. In addition, one need not test explicitly for inconsistency since it eventually results in computational problems. This is also a disadvantage in a sense, since no hint is given as to what aspect of the specification was inconsistent. This makes the high-level specification difficult to "debug".

5.2 Loose Ends

There are many aspects of planning that are not handled in this model as it now stands. Rather than build a planning system that does *everything* in some limited task domain, we tried to build up a general theory in order to model some of the most crucial aspects of temporal representation and planning. While it was impossible to model *all* aspects of planning that might arise, much thought went into some related areas of planning and how they might be included by extending the system.

A central issue in planning concerns the notion of *dependence*. In many planning theories, information is "directional". For example, there is a difference between "I must arrive at the meeting by the time it starts" and "The meeting will start once I arrive there". Even though both events, *arrival* and *start-meeting*, will occur simultaneously in the final plan, there is a directionality involved. In one, the meeting-time constrains the arrival time and in the other, the arrival time constrains the meeting time. This is represented in Allen's system through the distinct representation of (A before B) and (B after A). In contrast, our system is fully "bidirectional"; at the high-level of representation, the distinction can be represented syntactically, although, in the end, a simple bidirectional equality constraint is used. This is necessary to the notion of compromise in our system, represented through mutual dependence. For instance, in specifying "Do homework after lunch", one allows for the fact that a delay in lunch will *push* the start of *homework* forward in time, while at the same time, an excessive amount of homework can serve to *push* the end of *lunch* back in time to allow for more time for homework. It is possible to model *dependence* in our model as follows. Simply allow for different levels of planning; plan for the *highest priority* events first. Then specify exact constraints for the events as they were planned, using that plan as a *skeleton*, upon which the lower level may be planned.

A related problem concerns what is *really* meant by "fuzziness". In our system, a measure of fuzziness is related to the flexibility of a certain constraint because compromises are allowed. For instance the duration of a shower depends on how much of a hurry one is in. This duration is *flexible* to a certain extent. However, where compromise is not an issue,

"fuzziness" seems to be more an issue of "probability". For example, while the duration of a train ride might be described as "around three hours", this information gives uncertainty information, rather than a measure of flexibility. The best that a planner can do is to prepare for some worst case scenario. This can be seen to relate to the notion of *dependence* since the planner has no control over what will happen, and must simply "plan around" these occurrences. For example, where rain is predicted to start "around noon", one can not delay the rain in order to have a picnic. The best approach here is simply to generate a random number, using a probability curve based on the constraint: the more uncertain a specification, the broader the probability curve.

Anther crucial issue in planning concerns "disjunction". Many times a *choice* must be made as to how the plan is to progress. A choice between doing X on Monday and doing it on Friday does not imply that the best time to do X is on Wednesday. There are many levels at which disjunction may apply. At the lowest level, one may have a choice between an appointment at 1PM or at 2PM. At a higher level, one may have a choice between (A before B) and (B before A); the only constraint being that the time intervals are *disjoint*. Finally, at the highest level, one may have a choice in how a high level goal is to be divided into subgoals, as in the choice between eating dinner at home and going out. Currently, the only way to handle disjunction is to plan for each possibility and use the measure of the energy of the resulting spring system to tell which plan is best. Even here, there are complications since there may be *preferences* as to how plans are to be created, independent of how well the plan is timed out. For example, given a choice between *wash-face* and *take-shower* in the morning, most people would include *take-shower* in their plan, even if it meant they would have to rush later on.

Anther problem concerns "conditional planning". For example, a specification such as "brake the car if the traffic light is red" is strongly conditional, implying that the fact that the traffic light is red should somehow *activate* the braking of the car. Not only does this not fit into our system as it stands, but there may be times that conditional planning is not called for. For instance, in developing film, one must have it dark; however, if the film is *due* at a certain point in time, one can not simply wait until the room becomes dark. One must *act* to make that condition true if one is to have the film developed according to schedule. In addition, conditional planning would require some kind of daemon, watching for certain conditions, signalling some kind of *replanning* in the event that some crucial condition becomes true. Replanning would also be required as the animation progresses. since the passage of time makes the planning of events in the past *frozen*.

Another kind of planning not handled by our model concerns repeated action. Specifi-

cations such as "move the boxes, one at a time", "hit the nail repeatedly until the head is flush with the wood", and "stir occasionally" refer to noncontinuous time intervals which are nevertheless strongly related to each other. The best way to handle such specifications is probably to have one process which will *instantiate* the intervals with the appropriate relations to each other, while the planner can then be called to integrate these intervals with eact other and into the overall plan.

5.3 Implications

Finally, going in a new direction, there are many aspects of this system that can be extended to domains other that planning and temporal representation. The primary advantage of this planning system (and its underlying motivation) concerns the fact that it allows a great deal of freedom in the types of constraints expressible. The user can specify exactly what needs to be specified in terms relevant to the task at hand. The details of finding a *solution* to the set of constraints are left in the planner, where they belong.

While our system works with planning in *one-dimensional time*, the work of Witkin, et al [10], is concerned with spatial positioning in *three-dimensional space*. There work is similar in spirit to ours in that they can specify constraints in positioning for the construction of an object. By specifying the relevant characteristics of the relations between sub-objects, they solve for the best positioning and the object "falls-together" automatically. The common idea is that the user need specify only the relevant constraints while some other process computes a definite solution satisfying those constraints. Flexibility is increased as well: since modification is at the level of relevant description, a single aspect can be modified resulting in a repositioning conceivably changing the whole structure around.

Furthermore, there is no reason not to do the same for spatial placement in two dimensions, such as in the layout of a diagram or drawing. Often, in modifying a pictorial diagram, one must go to a great deal of trouble moving text around, resizing and redirecting relevant *arrows*, etc. If there were a "diagram specification language" in which specifications relevant to diagrams were allowed, one could specify the relevant features of such a diagram and a process would draw it up according to the specifications. In addition, any modifications would simply be a matter of modifying the specification. Fitting it additional boxes, circles, etc., into the plan would simply require the relevant additional specification, as opposed to a horrendous task of moving everything else out of the way.

By combining computational algorithms from *optimization theory* with ideas from artifi-

cial intelligence, we can, in principle, tailor the interface level to high-level specification and abstraction; while allowing the computer to figure out the "details" of how such ideas are to be implemented. In many diverse domains, one can imagine telling the computer *what* is required and allowing the computer to decide *how* to accomplish the task, rather than specifying *how* in exact detail, and never really getting what was actually intended.

5.4 Conclusions

We have presented a system in which temporal information is clearly and easily represented, while at the same time, an efficient algorithm can be used in order to *solve* the system. Through the analogy of a "spring system", we have shown that intuitive *fuzziness* in specification can be represented in terms of *constraint strengths*, that the "spring system" exhibits behavior similar to what we would want in a temporal planning system, and that the system can actually be solved for minimal energy, resulting in the best plan for the given constraints.

References

[1] J.F. ALLEN. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[2] J.F. ALLEN and P.J. HAYES. A commonsense theory of time. In *Proceedings of IJCAI 1985*, pages 528–531, IJCAI, 1985.

[3] J.F. ALLEN and P.J. HAYES. Moments and points in an interval-based temporal logic.

[4] WISMER D. and CHATTERGY R. *Introduction to Nonlinear Optimization: A Problem Solving Approach.* Elsevier Science Publishing Co., Inc., 1978.

[5] CHARNIAK E. and MCDERMOTT D. *Introduction to Artificial Intelligence.* Addison-Wesley, 1985.

[6] JEFFREY S. GANGEL. *Masters Thesis: A motion verb interface to a task animation system.* 1985.

[7] J. MALIK and T.O. BINFORD. Reasoning in time and space. In *Proceedings of 8th IJCAI 1983*, pages 343–345, IJCAI, 1983.

[8] DEAN T. and MCDERMOTT D. Temporal data base management. *Artificial Intelligence*, 32:1–55, 1987.

[9] S.A. VERE. Planning in time: windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5, No. 3. May 1983:246–269, 1984.

[10] A. WITKIN, Fleischer K., and Barr A. Energy constraints on parameterized models. pages 225–229, 1987.

A Specification Syntax

A.1 High-Level Syntax

AllSpec →	(Spec1 Spec2 Spec3 ...)
Spec →	(**start** TimeInfo) | (**end** TimeInfo) | (**duration** DurInfo)
TimeInfo →	(Fuzz-term Clock-time) | TimeSpec
TimeSpec →	([DurInfo] **after** TimeSpec) |
	([DurInfo] **before** TimeSpec) |
	(**start** Intv-label) |
	(**end** Intv-label) |
DurInfo →	(Fuzz-term Clock-dur) |
	(**more-than** Clock-dur) |
	(**less-than** Clock-dur) |
Intv-label →	*Interval Label* (such as Eat-Lunch)
Fuzz-term →	*Linguistically Fuzzy term* (such as *about*)
Clock-time →	*Clock Time* (such as 12PM)
Clock-dur →	*Clock Duration* (such as 10min)

A.2 Low-Level Syntax

Intv-Spec →	(Time-Spec Time-Spec Dur-Spec)
Time-Spec →	(Intv-label **start**) |
	(Intv-label **end**) |
	(Fuzz-term Clock-time) |
	nil
Dur-Spec →	(Fuzz-term Clock-dur) | nil
Intv-label →	*integer*
Fuzz-term →	*real*
Clock-time →	*real*
Clock-dur →	*real*

A.3 Examples from High-Level Syntax

We must plan a day for which the weatherman is predicting snow:

```
Blizzard:  ((start (around 7))
            (end (around 10))
            (duration (3 hr)))
```

Followed immediately by a period of light snow lasting one hour:

```
Snow:  ((start (end Blizzard))
        (duration (about (1 hr))))
```

Must shop for plenty of food beforehand

```
Shop:  ((end (before (start Blizzard)))
        (duration (about (1 hr))))
```

Sledding! - must start after Blizzard condition, maximize duration

```
Sledding:  ((start (after (end Blizzard)))
            (end (before (start Dinner)))
            (duration (pref (12 hr))))
```

Shovelling ... - must start after all snow has fallen

```
Shovelling:  ((start (after (end Snow)))
              (duration (about (2 hr)))
              (end (before (start Dinner))))
```

Hot-Chocolate - but only after done shovelling & way before dinner

```
Hot-Choc:  ((start (after (end Shovelling)))
            (end (more-than (2.5 hr) before (start Dinner)))
```

A.4 Examples from Low-Level Syntax

Class starts *close* to 10AM and ends *close* to 10:50AM. *Close* will mean about five minutes in this instance.
`(Class (2 10AM) (2 10:50AM) nil)`

This interval specifies the time taken to get to class. The start time is to be determined based on other parameters given. This interval should end at the start of class, and travel time can be estimated at about 45 minutes.
`(Go-Class nil (start Class) (10 45min))`

Theoretically, the time spent between *class* and *meeting* should start at the end of class and finish at the start of the meeting. It is known that it will take about 5 minutes to go between classes, maybe 3.5 if one runs.
`(Go-Meeting (end Class) (start Meeting) (10 5min))`

The meeting should start in the neighborhood on 11 o'clock with maybe 5 minutes on either side. In addition, it should end at 12 o'clock with maybe 3 minutes on either side.
`(Meeting (5 11AM) (2 12PM) nil)`

Lunch will start at the end of the meeting and can be assumed to last around 30 minutes, give or take about 10 minutes. Here *lunch* is assumed to be the interval of time *devoted* to lunch; hence, the time taken to go to the eating place, buy lunch, and eat it.
`(Lunch (end Meeting) nil (10 30min))`

B Decomposition of High-Level to Low-Level Syntax

If *time-point X* is to occur **before** *time-point Y*, create a dummy interval with start:X and end:Y.

If *time-point X* is to occur **after** *time-point Y*, create a dummy interval with start:Y and end:X.

If *time-point X* is to occur *duration-spec* **before** *time-point Y*, create a dummy interval with start:X, end:Y, and duration specification: *duration-spec.*

If *time-point X* is to occur *duration-spec* **after** *time-point Y*, create a dummy interval with start:Y, end:X, and duration specification: *duration-spec.*

If *time-interval A* is to last **more-than** *duration*, then create dummy interval B with start being the start of A and duration being exactly *duration.* Then create dummy interval C with start being the end of B and end being the end of A.

If *time-interval A* is to last **less-than** *duration*, then create dummy interval B with start being the start of A and duration being exactly *duration.* Then create dummy interval C with start being the end of A and end being the end of B.

C Technical Details

System Integration: Currently, the symbolic manipulation routines involved in compiling the *high-level* to *low-level syntax* is written in LISP, while the quadratic solution method is written in 'C'. As it has been tested, the LISP routines write out a file: *tempout*, which the 'C' routines read in, resulting in a print-out to the screen of the resulting timing.

Optimal Specification: The specifications easily handled consist of the "fuzzy specifications" and the "simple relations" between intervals. Overuse of specifications such as "exact specifications" and "before/after" type relations are much more likely to result in inconsistency. For example, specifying that an interval start at 1PM, end at 2PM, and last exactly two hours will result in overdefinition. Even if exactly one hour is specified as the duration, the specification is still overspecified by the algorithm's reckoning. This can all be avoided by introducing a bit of fuzziness into the specification such as "almost exactly" 1PM, 2PM, and one hour. *Exact* specifications should be used only when dealing with such

events as alarms going off and the such. Furthermore, before/after relations can easily result in underdefinition since it does not specify how much before or after. This leaves the system with a set of plans, all as good as each other, resulting in no definite solution. By introducing a small preference, one can avoid these problems somewhat. If *any* information can be given relating to *how much* before or after, this will allow the system to find a solution. Since it is a "very small" preference, it should not cause problems should conflicts arise. Also, it should be obvious that an interval be constrained in some way, else it will disrupt the entire solution. The best way to handle this problem is probably to use many defaults with "very weak" preferences.

Efficiency: The *compiling process* of converting high-level to low-level syntax is pretty straightforward and should be $O(n)$, where n represents the number of intervals. In the quadratic solution algorithm the matrices involved incur storage of $O(n^2)$ while the computation involves time of $O(n^3)$.