Representing and Propagating Constraints in Temporal Reasoning
Hany Tolba, François Charpillet and Jean-Paul Haton
CRIN-CNRS and INRIA-Lorraine
Campus Scientifique - B.P. 239
54506 Vandœuvre-les-Nancy Cedex
FRANCE
e-mail: tolba, charp, jph@loria.crin.fr

Abstract
We propose in this paper a new temporal representation combining the notions of intervals, dates and durations. The manipulation of this representation is based on the notion of time map managers (TMM) allowing both kinds of constraints, symbolic or numeric. These algorithms are a generalization of AC4 an optimal algorithm for arc-consistency and can handle n-ary constraints.

1 Introduction
The rising need for reasoning about time in various applications of artificial intelligence led researchers to propose different representation schemes for temporal information. A natural way to refer to a temporal event consists in making references to a clock providing a quantitative or numerical representation of time, as well as several concepts such as duration and calendar. However, a clock reference is not always available; in this case a qualitative or symbolic representation of time can be used to describe such a situation. When temporal constraints between two objects cannot be defined exactly, a disjunctive representation of these constraints can be used. In fact, the temporal description used in every day life usually involves both types of information.

The problem of temporal reasoning can be classified into two approaches. The first is based on the Operations Research (OR) where the only ambition is to place events and/or plans on a temporal reference line. We can mention the work of Vere [13] in the DEVISER system where he introduced the concept of window. Another interesting work is that of Tate and Bell [2]; the authors used a scheme based on upper-bounded and lower-bounded windows in their planning system. Both works are based on the longest path analysis technique to find a consistent plan. The second approach is based on the formalization of temporal reasoning over actions and establish causality relations by using temporal logic. Among them the reified logic is the most common one in the field and inspired from two major research works carried out by McDermott [6] and Allen [1].


Allen carried out a fundamental work using intervals to represent time [1]. He proposed a set of thirteen relations to state all the possible and mutually exclusive relations which can take place between two given intervals. Disjunctive relations between intervals are used to take into account the incompleteness of the world. A propagation algorithm eliminates all the disjunctive relations which imply inconsistencies. This process is managed by a Time Map Manager (TMM) which works on a network of constraints in which the nodes are intervals and the arcs disjunctive relations. A propagation algorithm calculates all the possible relations between the nodes and eliminates all the disjunctive relations which imply inconsistencies over the time map. In the same work, the authors presented a subinterval algebra which has the advantage of being complete and easily formulated using point algebra. Ghallab used this subinterval algebra to index a temporal database in a system for planning the movements of a robot [5]. Since the problem of the verification of the consistency of interval algebra is NP-complete [14], Allen [1] used an approximation to simplify the computation done by assuming only local consistency between three nodes of the graph. The complexity of such an algorithm is \(O(n^2)\). van Beek [12] proposed another algorithm to overcome the 3-summits problem but with a higher complexity rate \(O(n^3)\).

2 A New Approach
2.1 Introduction
We propose a new approach based on a discrete representation of time which allows to mix both quantitative (durations, start and end points) and qualitative (Allen’s 13 primitives) temporal information. The manipulation of both symbolic and numerical temporal information is done via two temporal constraints propagation levels. This original method of propagating
temporal constraints permits flexible control over both levels as well as the independence between those two levels.

2.2 Intervals with Dates and Durations

Most explicit time measurements use discrete values to represent time. We propose a discrete model of time which relies on the temporal reference Tr. Tr is the maximal set of discrete and adjacent temporal units ut (which can be considered as time points). Each unit represents the smallest discrete portion of time that can be obtained over the temporal reference; it is as well the basic unit for building any other basic temporal entity over Tr.

Using these units we can define another temporal entity, i.e., the interval. Any interval I can be represented as a subset \{uj\} of Tr. I is represented by a couple of units (dsup(l), dasup(l)), where dasup(l) and dasup(l) are the start and end time respectively where the precision depends obviously on the size of ut.

In this formalism the duration D(I) of an interval I is the cardinality of the subset of Tr defining it. Every interval has a life duration in which it must take place, represented by a temporal window. These windows can be easily integrated in our model and benefit from its discreteness to represent the uncertainty over the start and end bounds of an interval. A window W is a set of intervals defined as a temporal object represented by a triplet of units (dasup(W), dasup(W), D(W)), where D(W) is the duration of the intervals belonging to W. An instance of an interval taking place in a window is called an occurrence.

Events represent intervals that take place during windows. These events constraint themselves numerically using their durations and the windows during which they take place. The initial state for the universe in which we are working is to be represented in the following manner. Between any two events we can define any number of Allen's thirteen primitives. Every event is self constrained by its duration and the window in which it takes place. The propagation of constraints is used to deduce all the possible relations between any pair of events leading to a consistent labeling. The events which are numerically self constrained use the result of the propagation to eliminate the impossible numerical constraints represented in the windows in function of the existing symbolic constraints between events.

3 Propagating Temporal Constraints

3.1 Constraint Satisfaction Problem

Let \( X \) a set of variables \( x_i \), each defined on a discrete domain \{ai1, ..., aij\} and \( R \) a set of constraining relations on a subset of these variables. A constraint satisfaction problem (CSP) consists in finding all set of values \{aij\}, ..., aij}\} for \( x_1, ..., x_n \) satisfying all relations belonging to \( R \). The network \( G=(X,R) \) characterizing the CSP is generally an hypergraph in which the vertices represent variables and hyperedges represent relations. Since the CSP is NP-complete, algorithms assuming only local consistency were invented. These algorithms aim at transforming the network \( G \) into an equivalent and simpler one \( G' \) by removing from the domain of each variables all values that cannot belong to any global solution. A k-consistency algorithm removes all inconsistencies involving all subset of k variables belonging to \( X \). When k=2 and k=3 we say that the solution is respectively arc and path consistent. The k-consistency problem is in polynomial time \( O(n^k) \) where n is the cardinality of \( X \).

Among the arc consistency algorithms found in the literature, AC4 [8] developed in our laboratory has been proved optimal for discrete relaxation. This algorithm was extended in GAC4 [9] to handle n-ary constraints. This algorithm is also optimal. Both numeric and symbolic temporal reasoning can be viewed as a CSP. Therefore, it is useful to consider the results of these research topic in order to apply them in the field of temporal reasoning. We will now present the application to our symbolic and numerical temporal representation.

3.2 Symbolic Propagation

Allen has proposed a path consistency algorithm with a complexity of \( O(n^3) \). However, no proof of its optimality exists. Fortunately, it is possible to transform Allen's algorithm into a CSP. This algorithm relies on a graph \( G \) where nodes represent intervals and arcs are labelled by a set of Allen's primitives. For each triplet of nodes (ij,k) for which the admissible relations between (ij) and (ik) are known, the transitivity table allows the admissible relations between (ik) to be computed. As we have to find the actual relations between events, the relations should be considered as the labels by the propagation algorithm. \( G \) is thus transformed into another graph \( A(G) \) (cf. fig.1) in which a node noted (ij) represents the set of relations \{Rij\} constraining the intervals i and j. A node (ij) is linked with any node (ik) and (jk) by the hyperedges (ijk) of \( A(G) \). An hyperedge (ijk) specifies the following constraint:

\( \{Rij\} \circ \{Rjk\} = \{Rik\} \)

Figure 1. Transformation of Allen's graph into hypergraph for CSP.
3.3 Numerical Propagation

The numerical level propagates the numerical constraints represented by windows to find out the precise date of events. The numerical propagation consists in finding for each event all occurrences satisfying locally the temporal constraints (3-consistency). The symbolic constraints and the occurrences which cannot be satisfied are eliminated. Traditionally the propagation algorithm relies on a graph where nodes are events (with a domain in the form of a window) and edge are symbolic constraints. In order to transform this representation into a CSP one, a similar transformation as the one used for Allen's algorithm is needed.

Note that the occurrences domain of an event i belongs to a window noted Wi. Each occurrence of Wi is noted il. The nodes of the hypergraph noted (ij) represent the set of relations linking any couple (il, j) from Wi x Wj. A node (ij) is linked with any node (ik) by the hyperedges (ijk) of A(G). An hyperedge (ijk) specifies the following constraint:

\[ \forall il \in Wi, \exists j, \exists km \in Wk, \text{ such that } il \leq R_{ij} \leq j, \land il \leq R_{ik} \leq km \]\n
3.4 Two Level Constraint Propagator

Our temporal propagator works in two steps. First it tries to complete and deduce all missing information between events. Then, deduced information is used to reduce the size of the temporal windows and the symbolic relations linking every pair of events. The numerical and symbolic temporal constraints are handled by two separate communicating TMM. For these two levels of constraints propagation we have developed two general constraint propagator: an extended version of GAC4 and Angel [11] which are used in this paper at the symbolic and numerical levels respectively.

The symbolic and numerical TMMs operate in an interactive manner. The symbolic relations which cannot be satisfied by the numerical level are deduced and fed back to the symbolic propagation level via a communication module. Inversely, the symbolic relations which are eliminated by the symbolic level are communicated to the numerical level. This feed-back operation provides a solution refinement and a flexible method to control the problem reduction. This procedure ends when there are no more results to be processed.

3.4.1 GAC4

GAC4 handles relations such R(i,j,...,k) specifying the admissible labels for the nodes i,j,...,k. It is represented by an edge in a hypergraph: R(i,j,...,k) can be defined as the enumeration of all the p-tuples of labels admissible for (i,j,...,k): ((i,a),(j,b),...,(k,c)). GAC4 works as a recursive label pruning. When a label a has to be removed from the set L of admissible labels for i, all the p-tuples including a have to be discarded. When a p-tuple is discarded from a relation R, it may happen that this p-tuple be the last of R which was supporting a particular label. This label has thus to disappear, and so on. An efficient implementation of the algorithm is described in [9]. The algorithm, in fact, runs in two steps. The first step consists in building the data structure from the list of admissible labels for all the hyper-edges R. The second step prunes the labels which are not admissible.

Algorithm Evaluation

We have compared the complexity and the run-time efficiency of four algorithms: Allen's approximation algorithm, two algorithms proposed by van Beek and Gac4. In the general case, the complexity of Gac4 is \( O(n^3) \), where n is the number of nodes of (g). However, as Allen's relations are oriented, we have to deal with their symmetry, i.e. compute three labellings for each hyper-edge. The following table gives the complexity of the four algorithms (n is the number of graph nodes and m is the consistency order):

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>complexity</th>
<th>consistency order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allen</td>
<td>( o(n^3) )</td>
<td>3</td>
</tr>
<tr>
<td>van Beek (one to all)</td>
<td>( o(n^3) )</td>
<td>3</td>
</tr>
<tr>
<td>van Beek (all to all)</td>
<td>( o(n^4) )</td>
<td>4</td>
</tr>
<tr>
<td>GAC4</td>
<td>( o(n^{10}) )</td>
<td>m</td>
</tr>
</tbody>
</table>

These algorithms have been coded in Common Lisp and implemented on a Sun-3/60 workstation. We obtained the following run-times:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>number of nodes</th>
<th>time (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allen</td>
<td>10</td>
<td>44.66</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>187.37</td>
</tr>
<tr>
<td>van Beek (one to all)</td>
<td>10</td>
<td>86.78</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>101.80</td>
</tr>
<tr>
<td>van Beek (all to all)</td>
<td>10</td>
<td>44.79</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>351.93</td>
</tr>
</tbody>
</table>

For the sake of clarity, GAC4 is presented apart:

<table>
<thead>
<tr>
<th>number of nodes</th>
<th>number of hyper-edges</th>
<th>initialization (in seconds)</th>
<th>pruning step (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>18</td>
<td>18.00</td>
<td>0.60</td>
</tr>
<tr>
<td>40</td>
<td>30</td>
<td>25.08</td>
<td>1.44</td>
</tr>
<tr>
<td>75</td>
<td>55</td>
<td>37.97</td>
<td>3.69</td>
</tr>
</tbody>
</table>

GAC4 appears to be more efficient than the other algorithms. The pruning step, beside being optimal, is very fast and most of the running time is spent in the initialization step. Note also that these two steps are independent, which makes GAC4 of flexible use. Also, GAC4 gives us the possibility of choosing the consistency order in which the propagation is to be done.
3.4.2. Angel

We have designed a path consistency algorithm, called Angel [11] capable of propagating n-ary constraints. Angel runs on the graph which has been processed by Gac4. The principles demonstrated for Gac4 are thus also true for Angel: the final result is a path-consistent graph. Contrary to most existing constraint propagation algorithms, Angel is dynamic, i.e. it dynamically builds the graph on which it works by progressively processing the constraints one by one. Whereas the input of Gac4 is the set of non-admissible relations delivered by the initialization step, Angel receives admissible set of relations under the form of a set of all the possible occurrences for a given event. The constraints received by Angel are placed in an input-diary to be processed in the order of arrival. If the hyper-edge corresponding to the current constraint does not exist yet, it is created.

If the set of occurrences is modified for any of the edges of the current hyper-edge, all the hyper-edges having the edge in question in common are to be loaded into the input-diary in order to be examined later. When the input-diary becomes empty, Angel stops, waiting for a new input. The symbolic relations which cannot be satisfied are deduced and fed back to the symbolic propagation level via a communication module. This feed-back operation presents a kind of solution refinement allowing this one pass procedure to be repeated till no more possible solutions or problem reduction can be obtained.

**Algorithm**

```plaintext
while input-diary <> nil do
  hyperedge = pop(input-diary)
  verify-constraints(hyperedge)
  if modified(hyperedge) then
    push(input-diary, succ(hyperedge))
  end
end
```

where `verify-constraints` applies the constraints specified by an hyperedge, `modified` checks if the nodes linked by an hyperedge have been modified and `succ` returns all hyperedges having a common arc with a given hyperedge.

**Evaluation for Numerical Propagation**

Since there is a finite number of labels and hyper-edges, Angel is guaranteed to terminate. As for the complexity, the fact that Angel is a dynamic algorithm led us to consider it from the following point of view. We normally start with an empty input-diary and the algorithm stops working when this diary is empty or when there are no more hyper-edges to be processed. Thus the complexity problem is controlled by the number of hyper-edges formed from the triplet of nodes \((i,j,k)\) defining an interval. The number of updates for an arc \((ij)\) is limited, due to the limited number of intervals found in the temporal graph. Therefore, the possibility for an hyper-edge to return in the agenda is limited. The complexity cost in this case is bounded by the number of hyper-edges and it will be in the order of \(O(n^2)\) where \(n\) is the number of the graph nodes.

4 Conclusion and Future Work

In this paper we have proposed a new representation of time which combines both symbolic and numerical temporal information. We have also presented an original constraint propagation strategy based on two level propagation. Our constraint propagation algorithms have been proven to be of less complexity cost with respect to other algorithms proposed so far. The Angel Algorithm presents the new feature of being dynamic, thus allowing to be used with an inference engine.

Presently, we are working on two parallel aspects: the first is the improvement of the representation in order to be capable of presenting different time measurements. The second aspect is the improvement of our TMM and constraints propagation algorithms by the introduction of intelligent control and strategy.

**References**