# Temporal Reasoning in Sequence Graphs

**Jürgen Dorn**
Technical University Vienna,
Christian Doppler Laboratory for Expert Systems,
Paniglgasse 16
A-1040 Vienna, Austria
email: dorn@vexpert.dbai.tuwien.ac.at

**Abstract**

Temporal reasoning is widely used in AI, especially for natural language processing. Existing methods for temporal reasoning are extremely expensive in time and space, because complete graphs are used. We present an approach of temporal reasoning for expert systems in technical applications that reduces the amount of time and space by using sequence graphs. A sequence graph consists of one or more sequence chains and other intervals that are connected only loosely with these chains. Sequence chains are based on the observation that in technical applications many events occur sequentially. The uninterrupted execution of technical processes for a long time is characteristic for technical applications. To relate the first intervals in the application with the last ones makes no sense. In sequence graphs only these relations are stored that are needed for further propagation. In contrast to other algorithms which use incomplete graphs, no information is lost and the reduction of complexity is significant. Additionally, the representation is more transparent, because the "flow" of time is modelled.

## Introduction

In many AI applications reasoning about time is essential and therefore several techniques for the explicit representation and processing of time have been developed. Most of these techniques use graph theoretic models, with time entities as nodes and temporal relations as edges. The application area we have in view is the control of technical processes which involves planning, scheduling, monitoring, and diagnosis. In contrast to areas like NLP, special characteristics exist in this domain that require appropriate techniques, but may be used also to improve the processing.

Existing methods for temporal reasoning are extremely expensive in time and space, because general constraint propagation techniques are applied. In our approach, the characteristic of most real-time applications is considered. In these applications programs run for a very long time without interruption. A controlling program loops forever and some temporal constraints are used seldom and others more often. Moreover, in scheduling and planning we have to tackle uncertainty about the future, which implies the necessity to represent this uncertainty efficiently.

Before introducing the representation and the propagation based on this model, we show why temporal reasoning is useful in this application area and which objectives should be achieved with a new technique. Additionally, we discuss other approaches that have similar objectives.

Temporal reasoning is used to assure or to prove consistency between a set of temporal qualified propositions. If a proposition is added that is not consistent with the existing knowledge base, either the new proposition is invalid or some of the old propositions are wrong. This decision cannot be supported by temporal reasoning. It has to be decided with causal reasoning of some kind.

The temporal consistency mechanism is used for different tasks. In planning (Allen 1991) the inconsistency indicates that a chosen action is not appropriate for a given goal. Either the action is inconsistent with the goals or the set of propositions describing other actions and facts in the planning environment is inconsistent with the chosen action. It is also possible that a new goal is inconsistent with the knowledge base. This states that it is impossible to achieve this goal and replanning is needed.

In scheduling of production processes (Dorn 1991) temporal reasoning is used to represent temporal constraints like delivery dates, durations of operations, slack times, and the temporal description of process plans. Usually the inconsistency indicates that a resource needed is used by another operation at the same time.

In process control and diagnosis (Nökel 1989) temporal reasoning can be used to recognize deviations between the expected course of the process and that course that actually happens.

Another purpose of temporal reasoning is the computation of new knowledge. New knowledge about temporal constraints can be deduced with intersection and transitive conclusions. In planning, the sequence of actions can be deduced and the start times for actions can be computed. In diagnosis, a new hypothesis may be concluded or time-outs for supervision can be computed through temporal constraints.

One of the first described applications that has used some kind temporal reasoning was that of (Kahn & Gorry 1977). The system was not based on a graph theoretic model and was therefore more transparent for a user of

*appeared in the Proceedings of the 10th National Conference on Artificial Intelligence (AAAI 92), San Jose, CA, Morgan Kaufmann, pp 735–740, 1992.*

this system, but was also restricted to a small application area. A concept called before/after-chains was used, that influenced our idea for the propagation of intervals.

Most popular and also a basis for our representation is the model of (Allen 1983). Unfortunately, this model is not very transparent and does not show the "flow" of time, because every interval in the interval graph is uniform and all intervals are connected with each other. Moreover, the space requirements for the representation of the complete graphs and the time needed for the propagation is very high.

Often time point calculus instead of interval calculus is proposed to reduce the amount of work to achieve a consistent graph. In (Vilain, Kautz, & van Beek 1990) it was shown that the global consistency for the time point consistency is achievable in polynomial time, but this advantage must be paid by a lower expressiveness. In planning and scheduling a usual constraint is to rule out that two intervals overlap. In Allens model this is expressed by $I_1$ {<, m, mi, >} $I_2$, but in the time point calculus such a constraint cannot be expressed.

In order to reduce time and space requirements, in (Allen 1983) reference intervals were proposed. Since he has not given any rules on the generation of reference intervals, information may be lost in this model. Hence, in (Koomen 1989) rules were given to construct reference intervals automatically by a program. Here, a reference interval must contain its intervals and therefore no information is lost and the computation of the relation between two intervals that are part of different reference intervals is easier. However, for applications that we have in view, reference intervals are not the adequate representation, because a hierarchical representation is used sparsely.

In (Dechter & Pearl 1988) heuristic ordering for constraint graphs was proposed, to improve the general constraint satisfaction problem. Such a kind of ordering could be the "flow" of time. In (Ghallab & Alaoui 1989) an algorithm was proposed to order intervals temporally and they have detected that the propagation process can be sped up with this technique. Their model consists of two graphs: one graph with all intervals which can be ordered definitely and one graph with intervals that can not be ordered, because their relations are uncertain.

We will now present a model of representation and propagation that uses some of these ideas. We use the concept of "flow" of time in a graph theoretic model and obtain thus a kind of an ordered constraint graph.

## Sequence Graphs

We have mentioned that the uninterrupted execution of technical processes for a long time is typical for our applications. A controlling program loops forever but the involved intervals and their constraints may differ. Arranging intervals of the process along a time axis we obtain a figure that is stretched along a hypothetical time axis. The parallelism in the process is comparatively few in contrast to the amount of intervals over the whole lifetime of the process.

The following example is typical for a technical process. It is a simplification of a set of intervals from a scheduling expert system in a flow shop (Dorn & Shams 1991). The different processes described by intervals is a simplification of the treatments for one charge. The set of intervals and their temporal constraints can be interpreted as a process plan. In the following discussion we use only this process plan, but the reader should have in mind that a lot of process plans must be combined in order to get one schedule. Important temporal constraints will be between intervals of different process plans and therefore it would not make much sense to use a reference interval for a charge. The scheduling expert system has to combine approximately 200 process plans for one week.

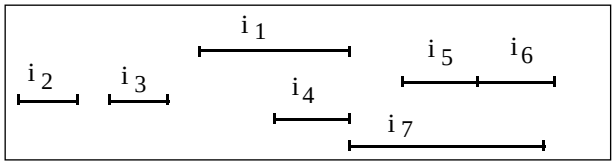


**Figure 1:** Process Plan described by Intervals

These intervals and their relations can be represented in a complete graph with 21 edges. Sequence graphs are based on the observation that a complete interval graph contains a path, where all intervals are constrained to occur one after another. This path is called sequence chain. Obviously, several chains may exist in one sequence graph. Applying sequence graphs, not every constraint is represented, because the transitivity property of the sequence chain is used. The complete graph of the process plan can be reduced to the following graph:

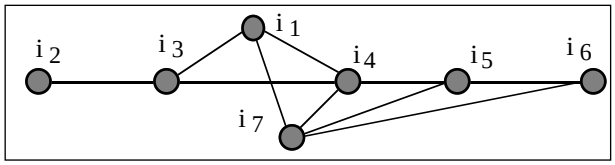


**Figure 2:** A Sequence Graph

One of the sequence chains emphasized by a bold line consists of the intervals $i_2$, $i_3$, $i_4$, $i_5$, and $i_6$. It is is a special subgraph with uniform edges. We can deduce, that $i_2$ is before $i_4$, because $i_2$ is before $i_3$ and $i_3$ is before $i_4$. No explicit transitivity rule is needed, because the relation is obtained from the position of both intervals in the chain.

The other intervals have to be connected explicitly to the sequence chain. But only relations to intervals which occur simultaneously have to be represented. The advantage of transitive chains is the reduction of edges in the graph and by that the amount of work and space. But we have to show that no information about the interval constraint is lost and every inconsistency is found.

In (Hrycej 1987) it was described how transitivity chains may be used to reduce the complexity of interval

algebra. He has used Allen's algorithm for transitive closure, but has changed the procedure "comparable". After the insertion of a new edge, superfluous edges are deleted. We improve this algorithm by deleting edges earlier. Thus, our algorithm is faster than that described by Hrycej. Furthermore, we use a stronger citerion to eliminate edges. Thus, we obtain also graphs with less edges.

## Representation

We represent temporal knowledge by intervals and constraints between these intervals. Sequence graphs are integrated into a tool called TIMEX (Dorn 1990) that uses Allen's relations. These are 13 mutually exclusive simple relations between intervals that are abbreviated by following symbols: =, <, >, m, mi, o, oi, d, di, s, si, f, fi.

Through disjunction of simple relations more complex relations can be formulated. These are interpreted as edges of an interval graph and they are represented as triples: $R = \langle I_1, C, I_2 \rangle$. To simplify theorems later on, we introduce predicates for some complex relations.

$$\text{unknown}(C) \leftrightarrow C = \{=, <, >, m, mi, o, oi, d, di, s, si, f, fi\} \wedge \text{unknown}(I_1, I_2) \leftrightarrow R = \langle I_1, C, I_2 \rangle \wedge \text{unknown}(R)$$

$$\text{sequence}(C) \leftrightarrow C = \{<, m\} \wedge \text{sequence}(I_1, I_2) \leftrightarrow R = \langle I_1, C, I_2 \rangle \wedge \text{sequence}(R)$$

$$\text{starts-before}(C) \leftrightarrow C = \{<, m, o, di, fi\} \wedge \text{starts-before}(I_1, I_2) \leftrightarrow R = \langle I_1, C, I_2 \rangle \wedge \text{sequence}(R)$$

$$\text{finishes-after}(C) \leftrightarrow C = \{>, mi, di, si, oi\} \wedge \text{finishes-after}(I_1, I_2) \leftrightarrow R = \langle I_1, C, I_2 \rangle \wedge \text{sequence}(R)$$

**Definition 1:** Complex Interval Relations

A transitive relation can be defined in a table as introduced by Allen. We use the function "trans($C_1$, $C_2$))" to denote transitive relations. The "unknown"-relation will be represented in graphs, but we can not generate further knowledge out of them. By truth tables it can be shown that the transitive relation of two relations is the "unknown"-relation, in case that one of them is unknown.

$$\text{unknown}(C_1) \vee \text{unknown}(C_2) \rightarrow \text{unknown}(\text{trans}(C_1, C_2))$$

**Theorem 1:** Transitivity of "unknown"-relation

An interval graph is a pair $\langle \Im, \Re \rangle$ consisting of a finite set of intervals $\Im$ and a finite set of interval relations $\Re$. All intervals of such a graph must be connected.

$$\text{intervalGraph}(\langle \Im, \Re \rangle) \rightarrow \forall \langle I_1, C, I_2 \rangle \in \Re \; [I_1 \in \Im \wedge I_2 \in \Im \,] \wedge \forall I_1, I_2 \in \Im \; [\text{connected}(I_1, I_2)]$$

**Definition 2:** Interval Graph

Two intervals are *connected* if a path exists between them in the interval graph. Such a path may contain also "unknown"-relations.

$$\text{connected}(I_1, I_2) \leftrightarrow \exists \langle \Im, \Re \rangle \wedge I_1, I_2 \in \Im \; [\exists \langle I_1, C, I_2 \rangle \in \Re \vee \langle I_2, CI, I_1 \rangle \in \Re \vee \exists I_3 \; [\text{connected}(I_1, I_3) \wedge \text{connected}(I_3, I_2)]]$$

**Definition 3:** Connection of Intervals

A sequence graph is an incomplete interval graph, because the properties of a sequence chain are used to reduce the number of edges in the graph. A sequence chain is a subgraph where all constraints are sequence constraints.

$$\text{sequenceChain}(\langle \Im, \Re \rangle) \leftrightarrow \forall \langle I_1, C, I_2 \rangle \in \Re \wedge \text{sequence}(C)$$

**Definition 4:** Sequence Chain

If an edge between two intervals $I_1$ and $I_2$ exists, then there is no knowledge about an interval $I_3$ that occurs between both intervals. Or with other words, there is no interval between two explicit connected intervals.

$$\text{sequenceGraph}(\langle \Im, \Re \rangle) \rightarrow \forall I_1, I_2 \in \Im \wedge \langle I_1, C_1, I_2 \rangle \in \Re \; \neg \exists I_3 \in \Im \; [\text{sequence}(I_1, I_3) \wedge \text{sequence}(I_3, I_2)]$$

**Axiom 1:** Sequence Graph

## Query for interval constraints

Since sequence chains are used the query for interval constraints is not so easy as in complete graphs. If the relation between two intervals that are not connected explicitly is asked, we must search for a path in the graph between the two intervals. However, this graph search is straightforward, because it must be searched only in one direction of the sequence chain. The relation could be only before or after.

Suppose we are looking for the constraint between the intervals $I_1$ and $I_2$. If there is no explicit edge between the two corresponding nodes in the graph we have two possibilities – $I_1$ is before $I_2$ or vice versa. If we have decided on one direction and this direction is right we never backtrack. The asked interval must be part of the sequence chain or it is connected explicitly to another interval in the sequence chain.

$$\text{before-in-sequence-chain}(I_1, I_2) \leftrightarrow \neg \exists \langle I_1, C_1, I_2 \rangle \wedge \exists I_3 \; [\text{before}(I_1, I_3) \wedge (\exists \langle I_3, C, I_2 \rangle \vee \text{before-in-sequence-chain}(I_3, I_2))]$$

**Theorem 2:** Searching Interval Relation

For planning or scheduling applications it seems to be a good heuristic to search into the future, because we suppose that the sequence chain into the past is longer for real-time application.

### Monotony in Sequence Chains

Some transitivity computations are not needed in sequence graphs, because no stronger constraint can be deduced. This property is called *monotony*.

$$\text{monotone}(C_1, C_2) \leftrightarrow \forall\, C_3 \\ \text{trans}(C_1, \text{trans}(C_2, C_3)) \subseteq \text{trans}(\text{trans}(C_1, C_2), C_3) \wedge \\ \text{trans}(\text{trans}(C_3, C_1), C_2) \subseteq \text{trans}(C_3, \text{trans}(C_1, C_2))$$

**Definition 5:** Monotony in Sequence Chains

In (Hrycej 1987) it was shown that two "sequence"-relations are monotone. The monotony states that the constraint via the two relations is always less constraining than successive constraint propagation via them. If a new constraint $C_3$ is added to a sequence chain $\langle C_1, C_2 \rangle$, it is not necessary to use the transitive computation via the sequence chain.

We formulate a stronger criterion, that will be used to reduce the edges in a sequence graph further. We only demand for a theoretical point instead of an interval to be between two intervals to make an edge between them superfluous. Suppose an interval $I_1$ is during an interval $I_2$ and it is known that interval $I_3$ is after $I_2$. Our theorem says that we must not represent the edge between the intervals $I_1$ and $I_3$.

$$(C_1 = \text{m} \wedge C_2 = \text{m}) \vee (C_1 \in \{<, \text{m}, \text{o}, \text{s}, \text{di}\} \wedge C_2 = <) \vee (C_2 \in \{<, \text{m}, \text{o}, \text{s}, \text{di}\} \wedge C_1 = <) \rightarrow \\ \text{monotone}(C_1, C_2)$$

**Theorem 3:** Strong Monotony in Sequence Chains

The theorem is provable by truth tables and it captures the property that was described in (Koomen 1989) that an interval that is during a reference interval must not be connected explicitly with another interval that is before or after its reference interval.

## Propagation in Sequence Graphs

The basic algorithm for propagation is similar to the algorithm for transitive closure in graphs described in (Allen 1983). A relation between two intervals is added to a graph and also to an agenda. All tasks of the agenda are performed in a loop. In the process of executing these tasks other tasks may be produced.

If a constraint is added to the graph we distinguish whether the new constraint is a sequence constraint or not. If it is one, some constraints are deleted and some are inserted. Otherwise, normal propagation is done.

For a task (an edge in the graph) a set of "comparable" intervals is generated. For every such interval the transitivity rule is applied in two directions. If the new computed interval constraint is stronger than the old one, the new constraint is added to the graph and results in a new task. The propagation algorithm terminates if no further task exists. The termination is safe, because tasks are only added when stronger constraints are added.

In the next subsection we describe how the set of comparable intervals is computed in sequence graphs. In the second subsection we suppose that an existing edge will be constrained stronger so that other edges can be deleted. In the last subsection the case is described that an edge to a new interval is added.

### Comparable Intervals

The function "comparable" computes all intervals for which the transitivity rule is applied. In the general case, these are all intervals that are connected explicitly with one of the intervals of the new edge. All intervals that are connected via an "unknown"-relation can be suppressed.

If the new constraint $C_1$ and the constraint $C_2$ that connects an interval $I_3$ with the new edge are monotone, then for $I_3$ the transitivity rule must not be applied.

$$\text{comparable}(\langle I_1, C_1, I_2 \rangle, \text{Set}) \leftrightarrow \\ \text{Set} = \{I_3 \mid \langle I_3, C_2, I_1 \rangle \vee \langle I_2, C_2, I_3 \rangle \wedge \\ \neg\, \text{monotone}(C_1, C_2)\}$$

**Theorem 4:** Comparable Intervals in Sequence Graphs

### Constraining two Intervals stronger

If the relation between two intervals is constrained stronger and the new constraint $C_1$ is a sequence constraint, two sets are generated. One set $\Im_1$ ("before-set") includes all intervals connected explicitly thru sequence constraints with $I_1$ so that they are before $I_1$ and the other set $\Im_2$ ("after-set") includes all intervals connected explicitly with $I_2$ thru sequence constraints so that they are after $I_2$. Suppose we have the following constellation:

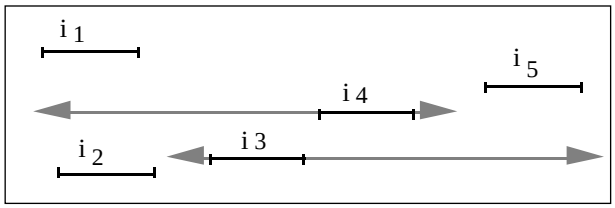


**Figure 3:** Constraining two Intervals stronger

The explicit constraints between these intervals are described in the following graph.

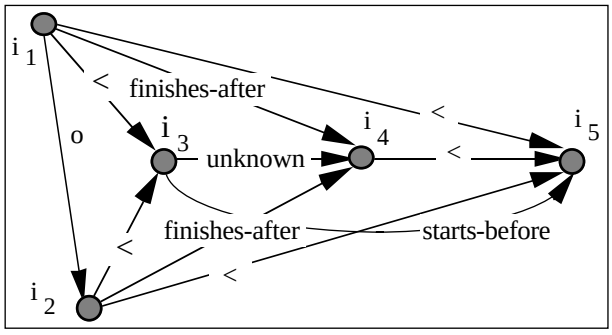


**Figure 4:** Sequence Graph

Suppose we constrain $i_3$ to be before $i_4$. Then $\Im_1$ consists of $i_1$ and $i_2$ and the $\Im_2$ of $i_5$. Now all edges between members of $\Im_1$ and $\Im_2$ and edges between $i_3$ and members of $\Im_2$ and between $i_4$ and members of $\Im_1$ are deleted.

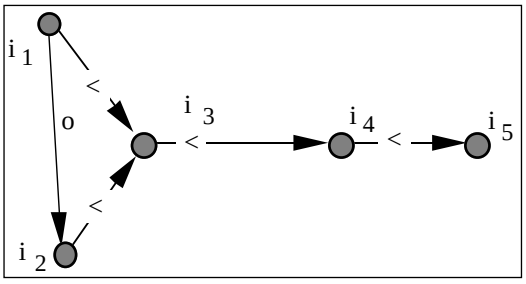


**Figure 5:** Reduced Graph

### Insertion of new Intervals

Now the case is described that a new interval is inserted and therefore new constraints are inserted. Suppose the following sequence graph exist:

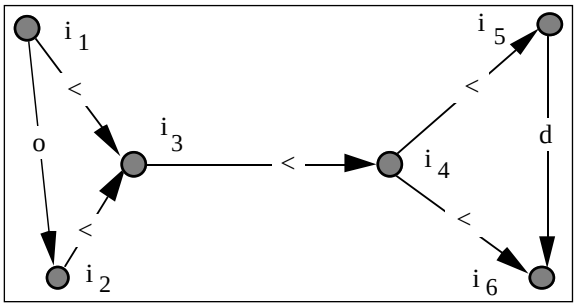


**Figure 6:** Insertion of a new Interval

If the interval $i_7$ is inserted and the interval $i_3$ is constrained to meet the new interval. Between $i_1$ resp. $i_2$ and $i_7$ no edges have to be added. But between $i_7$ and $i_4$, $i_5$ and $i_6$ "starts-before"-relations have to be established stating that these intervals starts after $i_7$. This interval constraint may be added without generation of new tasks, because we know that the interval constraints between the intervals $i_4$, $i_5$, and $i_6$ do not change. We get following graph:

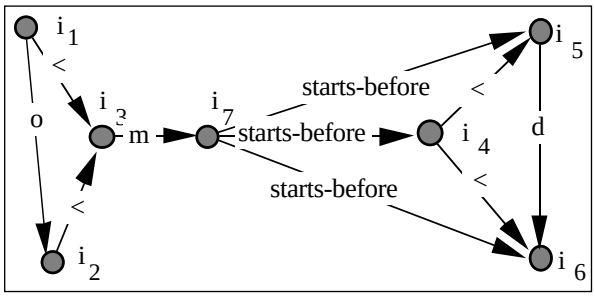


**Figure 7:** Insertion of a new Interval

If the new constraint would be a "before"-constraint nothing about the relations between $i_7$ and the other intervals could be concluded and therefore no further edges have to be added.

## Complexity

The complexity reduction thru sequence graphs is hard to define for the general case, but we assume special characteristics for our applications. In the next section we create a theoretical framework to estimate the reduction of complexity. In the second section we show the advantage on a more complex example.

### Theoretical Consideration

To examine the complexity of our new technique we define some new properties of sequence graphs. The *width* of a graph is the maximal number of concurrent intervals in the graph and correlates with the concept in (Dechter and Pearl 1988).

The *length* of a sequence graph is the length of the longest possible sequence chain. The length of a sequence chain is the number of intervals in the chain.

$$\text{sequenceGraph}(\langle \Im, \Re \rangle) \rightarrow$$
$$\max(|\ \Re\ |) < \text{width}(\langle \Im, \Re \rangle)^2 * \text{length}(\langle \Im, \Re \rangle) - 1$$

This formula is nice but the reduction depends heavily on the application. The width of a sequence graph can not be determined so easyly. Suppose all intervals of an application are arranged in an "overlaps"-chain, then all intervals could be concurrent.

### Practical Considerations

To get a feeling for the reduction of complexity in technical applications we take a scheduling example from the application described in (Dorn & Shams 1991). The first digit of the intervals indicate the number of a charge.

The constraints for one charge are: $i_{x1}$ m $i_{x2}$ m $i_{x3}$ m $i_{x4}$ m $i_{x5}$ m $i_{x6} \wedge i_{x7}$ m $i_{x5}$. The following constraints can be deduced by propagation with sequence graphs: finishes-after($i_{x7}$, $i_{x1}$) $\wedge$ finishes-after($i_{x7}$, $i_{x2}$) $\wedge$ finishes-after($i_{x7}$, $i_{x3}$) $\wedge$ $i_{x7}$ {=, f, fi} $i_{x4}$. These are ten constraints. In a complete graph we would have 21 constraints for one process plan.

One important constraint connecting process plans demands that two charges that use in the first treatment the same resource are scheduled after another. We introduce the interval $i_{x0}$ which describes the delay between two charges. Furthermore, we assume that the last treatment must be again sequential. So we constrain $i_{15} < i_{25} < i_{35} < i_{45}$.

Now we have 31 intervals connected in one sequence graph. We have inserted 30 constraints. The propagation algorithm deduces 143 constraints. The amount of constraints in a complete graph would be 32 * (31-1) / 2 = 496.

Figure 8 describes the intervals of four charges.

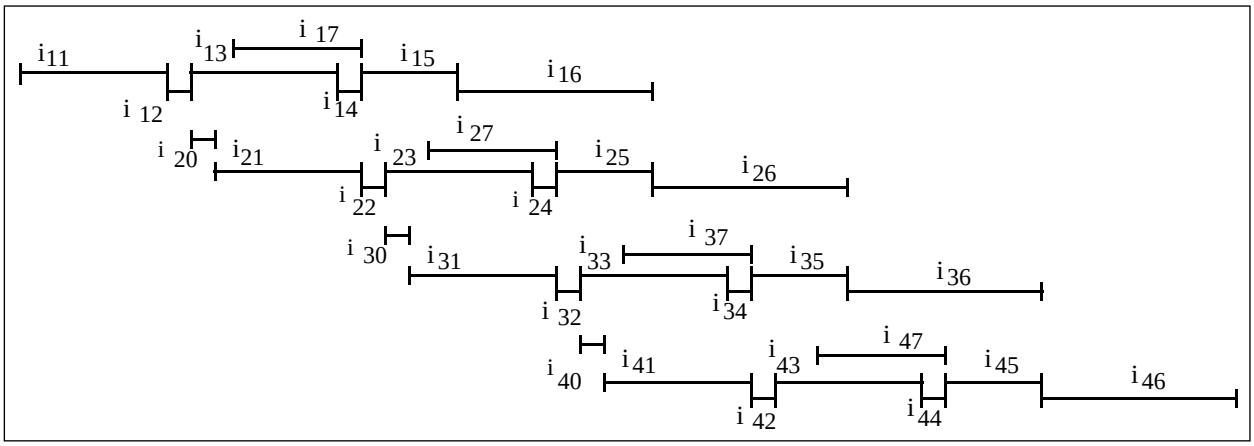


**Figure 8:** Practical Example

## Conclusions

We have presented an improved technique for temporal reasoning that is less complex in time and space than other known techniques. In contrast to other algorithms no information is lost and in contrast to time point calculus the expressiveness of Allen´s interval calculus is preserved.

The amount of reduction in our model depends heavily on the application, so that we can not make an universally valid estimation. But if we suppose that the average of concurrent intervals is l and the length of the longest chain is k intervals we estimate 2lk edges. For a complete graph it would be n * (n - 1) / 2 edges for n intervals. Unfortunately, if all constraints given are “overlaps”-relations, we will obtain no reduction of space.

Our method has one disadvantage. If there is a query for a constraint we have to search for a path between both intervals. In the complete graph we would find this constraint immediately as a constraint.

We have tested our representation with a number of examples from real world scheduling problems and in most cases a significant reduction of time and space requirements were determined in comparison with pure interval calculus. The reduction occurs preferably with large amounts of intervals.

Nevertheless, for most technical applications the presented model is insufficient, because also quantitative representation and propagation of time is needed. Furthermore, concepts for inexact reasoning with quantitative time is necessary and this reasoning must be combined with the qualitative reasoning. In (Dorn 1990) we have integrated both, and the reasoning from qualitative to the quantitative representation is quite easy and can be performed with linear effort. Unfortunately, we have not yet found good algorithms to conclude from quantitative to qualitative representation.

## References


Allen, J. F. 1983. Maintaining Knowledge about Temporal Intervals. *CACM*, 26(11): 823–843.

Allen, J. F. 1991. Planning as Temporal Reasoning. In *Proceedings of the 2nd KR*, Cambridge, Massachusetts, 3–8.

Dechter, R.; Pearl, J. 1988. Network-Based Heuristics for Constraint-Satisfaction Problems. *Artificial Intelligence* 34, 1–38.

Dorn, J. 1990. TIMEX – A Tool for Interval-Based Representation for Technical Applications.In *Proceedings of the 2nd Conference on Tools for Artificial Intelligence*, Washington, DC, 501–506.

Dorn, J. 1991. Qualitative Modeling of Time in Technical Applications. *Proceedings of the International GI Conference on Knowledge Based Systems*, München, Springer Verlag, 310–319.

Dorn, J.; Shams, R. 1991. An Expert System for Scheduling in a Steelmaking Plant. In *Proceedings of the World Congress on Expert Systems*, 395–404.

Ghallab, M.; Alaoui, A. M. 1989. Managing Efficiently Temporal Relations Through Indexed Spanning Trees. In *Proceedings of the 11th IJCAI*, Detroit, 1297–1303.

Kahn, K.; Gorry, G. A. 1977. Mechanizing Temporal Knowledge. *Artificial Intelligence* 9, 87–108.

Koomen, J. A. G. M. 1989. Localizing Temporal Constraint Propagation. In *Proceedings of the 1st KR*, Toronto, 198–202.

Hrycej, T. 1987. An Efficient Algorithm for Reasoning about Time Intervals. In *Proceedings Expertensysteme ´87*, Nürnberg, Germany 327–340.

Nökel, K. 1989. Temporal Matching: Recognizing Dynamic Situations from Discrete Measurements. In *Proceedings of the 11th IJCAI*, Detroit, 1255–1260.

Vilain, M.; Kautz, H.; van Beek, P. 1990. Constraint Propagation Algorithms for Temporal Reasoning. *Readings in Qualitative Reasoning about Physical Systems*, Daniel S. Weld, Johan de Kleer, 373–381.