ESPRIT 2434

FINAL IMPLEMENTATION REPORT

T I P :

Temporal Inference Propagator and Processor

Jörg - Ingo Jakob

Philips GmbH Forschungslaboratorien
Forschungsabteilung Technische Systeme Hamburg

Note: the two product descriptions "TIP (Software)" and "TIP (Hardware)" have been collected into one for convenience.

INDEX

1. Product Description and Motivation

Scheduling is the allocation of resources over a period of time in order to accomplish some tasks [Philips 1989]. Scheduling is distinct but not independent of planning, which is the determination of which tasks to be perform. All tasks must be sequenced so that a time ordering is imposed on them. We will refer to scheduling as encompassing scheduling, sequencing and planning, unless stated otherwise. Scheduling has gained a lot of attention recently, especially in industrial manufacturing. There are a number of factors which make scheduling a difficult problem to solve. The most prominent feature of scheduling problems is their combinatorial explosiveness meaning that the number of possible schedules grows exponentially along each dimension such as operations, machines, tools, orders etc. Nevertheless a powerful scheduling facility is required for almost all manufacturing enterprises. This facility is the Temporal Inference Propagator and Processor (called "TIP Software" and "TIP Hardware" throughout this contribution) which we describe in the following. TIP is a domain-independent temporal inference mechanism with non-temporal generalizations. We will show how TIP can been successfully applied for scheduling in the production domain.

2. Requirements Specification

This chapter is divided into two parts, mirroring the facts that (1) TIP as a temporal inference engine is independent of any application domain, but that (2) in the context of ESPRIT 2434, TIP found its application in the production planning domain.

2.1 General Requirements of Scheduling Systems

One feature of every scheduling problem is its set of constraints. Identification, representation, and correct and efficient use of constraints to prune the space of possible schedules are of overall importance. The set of all constraints defines the solution space in which a solution has to be found. In case of production scheduling, many constraints are simply temporal constraints. A scheduling system in the production domain has to be able to represent the following time specifications (examples):

- exact specification: event A occurs at 10:24, or event A occurs 21 secs before event B
- uncertain specification: event A occurs 1 to 2 hours before event B
- qualitative specification: proces A occurs during process B
- incomplete specification: process A occurs before or during process B
- quantitative specification: process A lasts 1 hour, or process A lasts 1 to 2 hours longer than process B.

TIP provides representation mechanisms for all these type of temporal specification: exact, uncertain, qualitative, incomplete, quantitative specifications. In addition, extensions towards general constraint representations are included in TIP (simple arithmetical constraints; local optimizations; second order constraints, also called demons or if-then rules).

2.2 Scheduling Requirements of a CIM Environment

TIP is an inference engine for temporal logic which is to be used as a component in a larger system. Our application domain is production planning in a CIM environment. The knowledge-based CIM environment is shown in Figure 1.

Explanations: TIP operates in a CIM environment, a fact which is represented by the three components factory, CIM planner (for example, the job shop scheduling expert system AIPLANNER) and user interface. However, TIP is capable of operating in any other domain which uses time inferencing: TIP is domain-independent. There typically are two data formats which are incompatible: the application domain, and the temporal inference domain. (Both domains are separated by a dashed line in Figure 1). The data format of TIP is termed "situation" format. The modules domain-to-situation translator and situation-to-domain translator convert one data representation to the other.

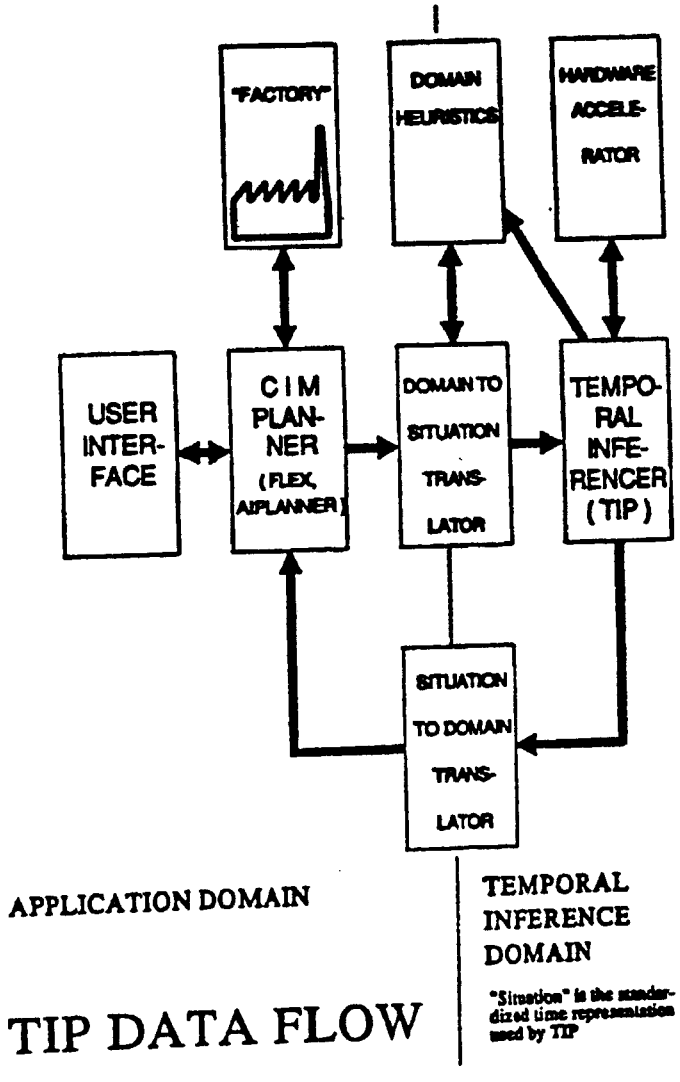


Figure 1: System Concept for an Advanced CIM/AI Controller for Production Planning

The CIM planner provides an intial situation to TIP. However, this initial situation is too general ("under-constrained"), containing ambiguities, thus defining a whole solution space. The task of TIP is to explore this solution space, and to find the best solution according to some evaluation criteria.

The "exploring" modules are TIP (Software) and its optional hardware accelerator TIP (Hardware): a situation is constrained *step by step*, resulting in one, a few or many situations where each single situation is fully constrained (not under-constrained as the initial situation was). These fully constrained situations are instantiations of the initial situation as provided by the CIM planner. They are commonly referd to as "solutions". As a common property, these solutions do not contain any temporal ambiguities: each time interval is accurately positioned on the time line. Converting these solutions back to a data format understandable to the CIM planner is the tasks of the situation-to-domain translator.

Experience with temporal inference indicates that it is necessary to heuristically prune solution space (see below). To a large extent, these heuristics depend on the application domain. This is the task of module domain heuristics. A second task is to implement a control loop which enables a guided search algorithm (diagonal arc). This control loop uses the evalution criteria for solutions found (as mentioned above). Together with the hardware accelerator module, it implements an efficient temporal reasoner TIP (e. g., Temporal Inference Propagator).

Hardware implementation: TIP (hardware acronym: Temporal Inference Processor) and its associated modules will be kept on one hardware platform (currently, IBM PC-AT compatible computers). The software modules of the application domain may reside on any other host. Communication can be done using a local or wide area network [cf. the product description of the Portable Communication Architecture (PCA) in this volume]. This way, TIP can use CIM planner as a data base. CIM planner delegates a temporal planning task to TIP, and the domain-to-situation translator assembles all the necessary data from CIM planner. Finally, the results are sent to CIM planner by the situation-to-domain translator. This is an example of knowledge-based distributed CIM processing.

3. Organization Structure

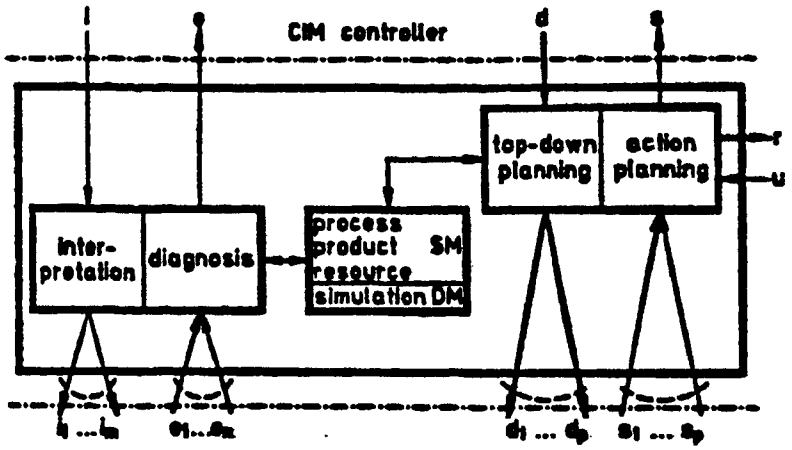
As a planning component, TIP is not restricted to any particular level in the GRAI decision grid. As an example, it has been applied to the workshop and workcell levels of the production management system (Figures 1 and 2; expert systems cooperation with AIPLANNER [Jakob 1991a]). TIP is intended to be an integral part of the production management system and of the CIM/AI controller, its task being temporal and common sense inferencing on standardized decision frames exchanged ([Bünz 1989], [Parr 1991] and Figure 1).

Horizon / Period \ Function		Operational Management	
		Production	Maintenance
AUDIO SHOP	1 month 3 days	release workcell orders	determine maintenance intervals
		d_0 ↓	
SMD WORK-CELL	3 days 1 shift	schedule workstation jobs	r_m ← schedule preventive maintenance
		↓	
11 WORK-STATIONS	1 shift 1 hour	d_5 perform jobs	perform maintenance jobs

Figure 2: GRAI Decision grid for TIP application example

4. Coordination Structure (CIM Controller)

The structure of the CIM/AI Controller is based on the standard ESPRIT 2434 software architecture (Figure 3). The modules are to be found in Figure 1 as well.



Models:

SM	static models
DM	dynamic model

Decision Flow:

d	decision frame from upper level
d_1-d_p	decision frame to lower level
r	request from/to same level

Information Flow:

e	status of resources to upper level
e_1-e_n	status of resources from lower level
i	status of upper level CIM module
i_1-i_m	status of this CIM module to lower level
s	decision frame status to upper level
s_1-s_p	decision frame status from lower level
u	request status from/to same level

Figure 3: The Architecture of the CIM production module

5. Decision Processes and Decision Nets

Not available. TIP is a temporal inference engine. Although the underlying inference engine could be formulated in GRAI notation[1], we refrain from doing so, for there is no direct relation to the field of Computer Integrated Manufacturing.

[1] TIP's temporal inference engine has much in common with Prolog inference engines, in that it uses uninformed backtracking (TIP: also informed backtracking) when examining the solution space. Thus, the task of formulating TIP in GRAI notation would be similar to formulating a Prolog inference engine in GRAI net notation.

6. Representation and AI Modules

In order to motivate our temporal-logic based approach in scheduling, some drawbacks of typical project-scheduling methods (CPM and PERT) will be described.

CPM (Critical Path Method) has no provision to model the uncertainties which are essential in the manufacturing domain. For example, an activity which is expected to take 5 days to be performed, but might vary from 4 to 6 days would be treated not differently from an activity which is expected to take 5 days, but might vary from 1 to 10 days [Moder 1983]. It is evident that this behaviour is not suitable for the scheduling domain where most production processes are influenced by statistical fluctuations.

PERT (Program Evaluation and Review Technique) uses a statistical approach to describe uncertainty. Each activity is characterized by three values: an optimistic time, a most likely time, and a pessimistic time. Based on these estimates, probabilities are derived that a schedule is finished before or after a specified end date.

The basic CPM and PERT procedures which produce a detailed schedule are limited in the sense that resource availablitities are not considered in the scheduling process. These procedures implicitly assume that available resources (personnel, equipment, material) are unlimited and that only technological requirements constrain the start and end dates. Therefore they often have been termed 'time-only' methods. One consequence is that schedules produced may not be realistic when resource constraints are considered.

Most domain-independent planning systems using AI techniques (e. g. NOAH [Sacerdoti 1977] and MOLGEN [Stefik 1981]) do not represent time explicitly. To cover that representational deficiency several approaches have been attempted, e. g. MOLGEN uses different object names to refer to the same object at different times in the plan.

However, previous work in knowledge-based production scheduling [Fox 1983] [Collinot and Lepape 1987] [Smith et al. 1986] [Elleby et al. 1988] has recognized the importance of exploiting temporal knowledge as a basis of finding feasible production schedules. Using a temporal logic instead of the traditional methods of Operations Research leads to the following advantages:

- temporal dependencies are made explicit;
- quantitative dates are not necessary at first. One can specify qualitative precedence and can later on add the necessary quantitative data as required (least commitment approach);
- separation of domain-specific and domain-independent temporal knowledge is possible and convenient (cf. Figure 1).

In representing temporal relations we can distinguish between two kinds of representations: symbolic representation for qualitative relations and numeric representations for quantitative relations. These two representations are both integrated into TIC. The numeric representation is used to refine the relations expressed by the symbolic representation and vice versa.

The qualitative inference procedure we developed is mainly based on Allen's temporal interval logic [Allen 1983]. It comprises 13 primitive temporal ordering relations to describe temporal dependencies between time intervals during which situations take place (Figure 4).

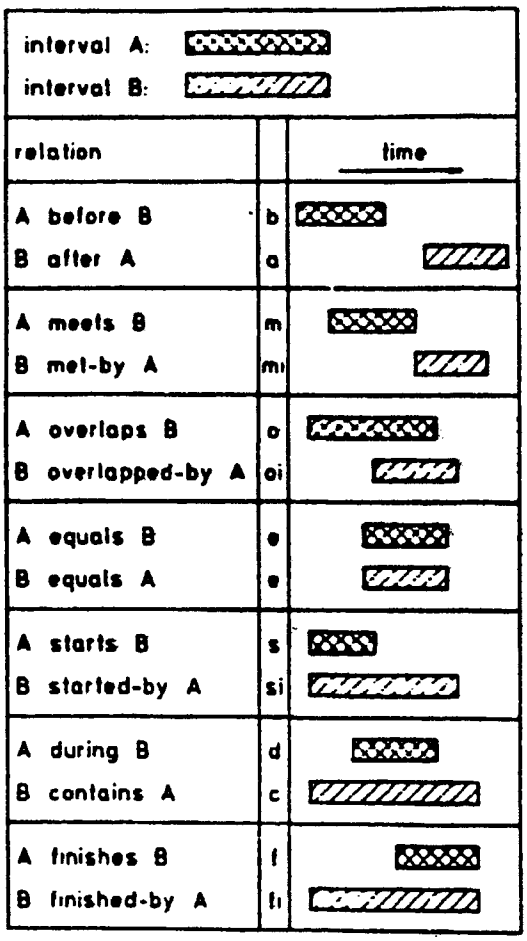


Figure 4: Representation of qualitative temporal data (13 primitive temporal relations)

The temporal relations are constraints as they constrain the qualitative order of the intervals relative to one another. Inferences take place by propagating these constraints between more than two intervals. The temporal knowledge which is expressed by the time intervals during which an assertion holds, and by the temporal constraints between them, is organized as a network. The intervals are the nodes and the relations the arcs. Allen does not use numeric data like the beginning or duration of an interval, but sticks to a purely qualitative description.

Example: "X (before, after) Y" means that either X must take place BEFORE Y, or X must takle place AFTER Y (exclusive or). Overlapping or containments are not allowed. This small example can be used to model that the two processes X and Y are not allowed to use the particular resource simultaneously.

All intervals are connected by these relations thereby constraining their possible sequences. The relations are mutually exclusive. In a consistent solution we can only have one relation at any one time. Therefore, after all qualitative relations have been specified they need to be propagated and made unambiguous in order to find consistent solutions. This process is called temporal inference.

Finding a solution by temporal inferencing is a classical constraint satisfaction problem [Stefik 1981] [Montanari and Rossi 1988] [Nadel 1988] [Haralick and Elliot 1980].

Up to this point we have described solely the qualitative relations. To compute the final production plan, a quantification step is necessary. For the representation of quantitative data of production plans we have adopted the notion of 'windows'. A windows of any time interval is bound by the six independent variables earliest start date, latest start date, earliest end date, latest end date, minimum duration, maximum duration. These data describe all possible positions of a window thus defining its set of possible occurences (SOPO [Rit 1986]).

Since these six variables do only partially depend on each other linearly, the graphical representations of a SOPO is done best in a two-dimensional coordinate system (Figure 5). The used abbreviations are: FAZ = earliest start date, SAZ = latest start date, FEZ = earliest end date, SEZ = latest end date, MIN = minimum duration, MAX = maximum duration. Using these six variables, the uncertainty on the exact occurence of an event can be expressed in a natural way. This is useful for a least-commitment approach: decisions can be deferred as long as possible.

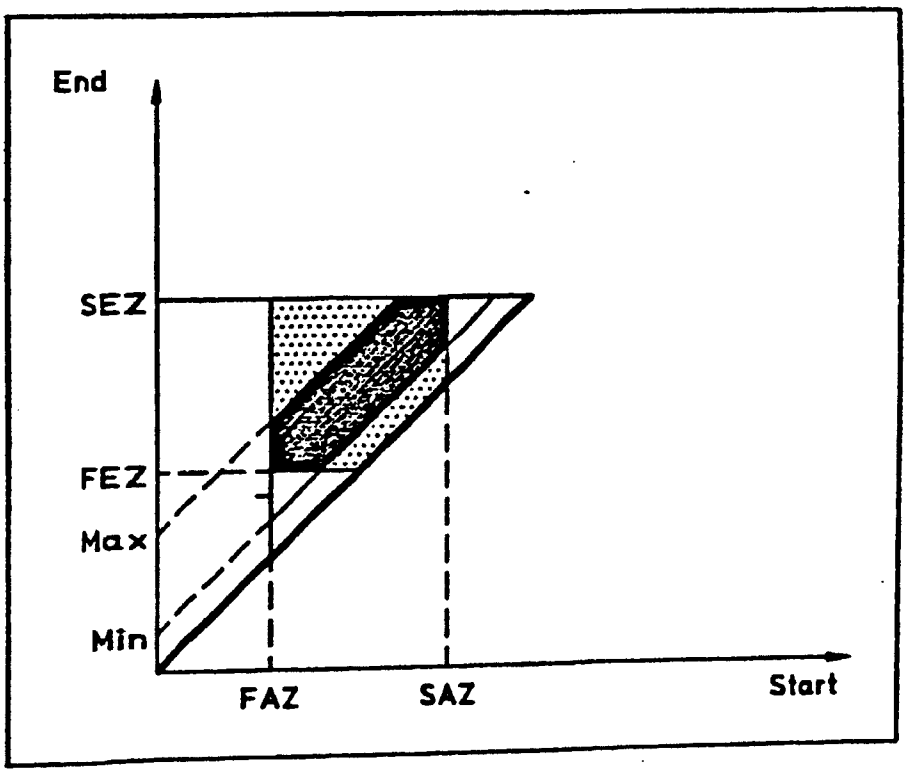


Figure 5: Representation of quantitative temporal data

7. Models

Not applicable. The TIP system and representation of temporal entities is domain-independent. There are no domain-specific models.

Special note: Of course, the domain-to-situation translator and the situation-to-domain-translator of Figure 1 accomplish the translation from a model-based space (e. g., production planning) to a domain-independent space. For the AIPLANNER example [Jakob 1991a], we can apply the Order Model as described in [Jakob and Isenberg 1991].

8. Inference

8.1 Problem Solving Architectures

TIP operates on a temporal knowledge base which is divided into a domain-dependent and a domain-independent part (Figure 1). The domain-independent part contains the knowledge of how to efficiently propagate temporal relations; the domain-dependent part contains the defined manufacturing situations and additional quantitative and qualitative constraints.

The temporal knowledge of TIP is expressed by so-called "temporal qualified assertions" (TQAs). A TQA states the validity of an assertion during a time interval. TIP accepts such temporal qualified assertions as input and propagates the effects. Inferences are drawn by propagating temporal relations according to the underlying transitivity (or consistency [Swain and Cooper 1988]) tables.

Using an interval logic is computationally expensive. Building consistent networks of TQAs is proved to be NP-hard in [Villain and Kautz 1986]. To decrease cost, Allen's algorithm 'ToAdd' uses only 3-consistency as an approximative method in establishing consistent temporal networks: the constraints between any three intervals ("triangles"; "path consistency") are guaranteed to be consistent. In a completely connected temporal net there are at most n * (n-1) * (n-2) / 6 such triangles. Allen's 'ToAdd' consumes O(n ** 3) time and O(n ** 2) space complexity.

To make temporal constraint propagation feasible, it is necessary to reduce the computational costs of this algorithm. To name three possible ways: by using so-called "refemce intervals", by using a heuristic-driven search mechanism, and by introducing non-temporal constraints (arithmetical constraints, second-order or meta constraints (if then constraints)). These approaches are described below and in [Jakob 1991a] [Becker 1991].

The first approach uses "reference intervals". A reference interval is an ordinary interval but it is used to cluster other intervals which are related by temporal proximity. These other intervals form the "group" of that reference interval. Each interval of a group can in turn form the refemce interval of another group of intervals, and so on (Figure 6). Since the costs heavily depend on the number of constraints between intervals, we tend to use reference intervals and restrict ourselves to the use of tree-like nets.

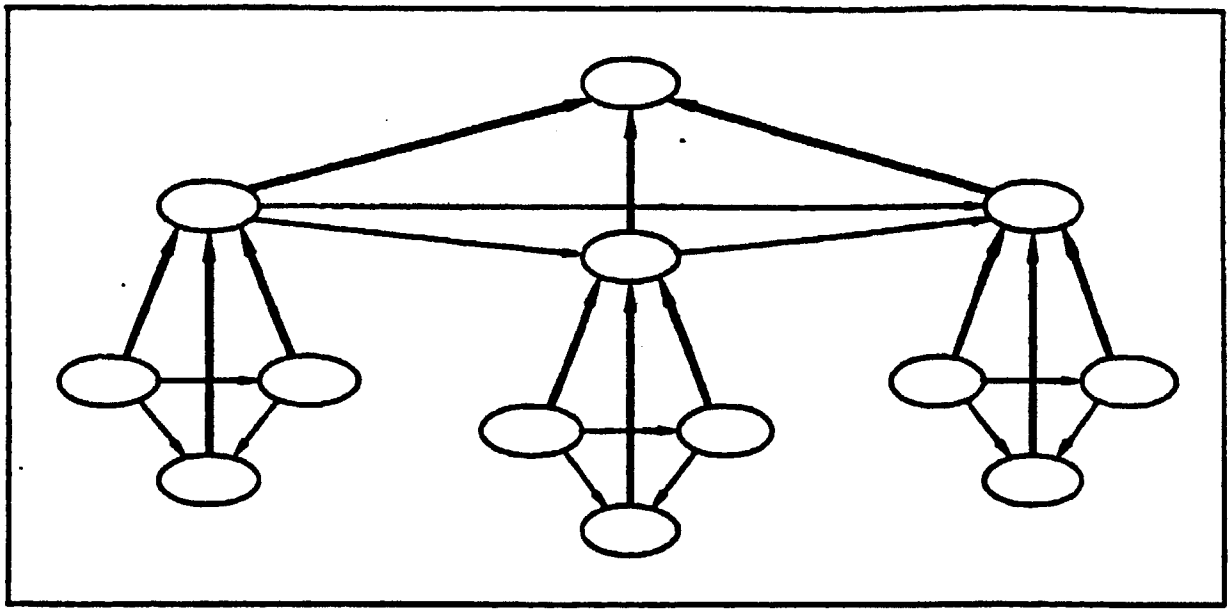


Figure 6: Temporal Net with Reference Intervals

The number of relations within a balanced tree-like net is typically O(n) = ((r ** height(tree)) - 1) * (r + 1) /2). Forbidding intervals to have more than one reference interval and keeping the tree-like structure strictly does not only prevent the net from 'flattening out'. It further reduces the general graph search to a tree search. This search is performed in O(p * height(tree)), with p depending on the position of the intervals within the tree-like net.

Further considerations concerning time and space complexity of TIP are to be found in Chapter 8.3 (Complexity).

8.2 Control Strategies and Meta Rules

Temporal inference can be implemented in two steps. The first version uses a depth-first search algorithm which generates all feasible solutions by expanding the temporal constraint net and finding all unambiguous expansions. The user has to assess this set of feasible order sequences to choose the desired solution, according to some quality criterion. This algorithm is completely domain-independent but computationally expensive. A second version uses extensions to depth-first search, such as best-first search (looking for only one solution), or intelligent search (using information which became apparent during the search process).

Depth-first search was described in Chapter 8.1 (Problem Solving Architecture).

Best-first search or looking for one solution is guided by heuristic rules (shortest processing time, earliest due date, first-in-first-out etc.), as well as user-defined priorities on any relation (minimize setup time by keeping similar orders together), in order to find the (nearly)

best solution. Furthermore the user can define priorities on any relation (e. g. by keeping orders together to minimize setup times).

We can assign relative priorities to the constraint net thereby directing the way inference proceeds. Possible priority rules are:

- make that relation unambiguous which promises to maximally prune solution space (first fail principle [van Hentenryck 1989]; this is a general strategy which holds also for depth-first search);

- make relation unambiguous which currently has the largest weight;

- relative importance of one primitive relation is weigthed against all other primitive relations of the same (compound) relation.

The determination of one order sequence starts with declaring temporal intervals, relation between these intervals, time windows per interval and priorities per relation. This is done using the TIP language as defined in [Jakob 1991a]. These declarations are then expanded into a network of objects representing temporal relations and intervals. To restrict the search of qualitative consistent solutions, those qualitative temporal relations are eliminated which contradict the quantitative time windows. In case a qunatitatively consistent solution exists, the algorithm terminates. Otherwise the second-best set of priorities is tried, the third best ,,. and so on. This generate-and-test approach uses domain and application specific knowledge to find a good solution which fulfills all given constraints. Under certain conditions, this procedure might fail and terminate without solution, for no complete traversal of the search space was performed.

Using the second version or approach we loose domain-independence but gain a lot of processing speed.

8.3 Complexity

8.3.1 Some Formal Definitions

Let N be the number of nodes (i. e. , temporal intervals) in a temporal net.. Let E = N * (N-1) /2 be the number of edges in a temporal net [there is one edge between any two nodes in a fully connected net]. Let there be a '<' ordering amongst all nodes and edges. Let T = N * (N-1) * (N-2) / 6 be the number of "triangles" in a temporal net [a triangle being defined as any triple of three edges (Ei, Ej, Ek), where always Ei < Ej < Ek]. Let S be the number of primitive relations of the underlying temporal logic (Allen's logic has S = 13).

Definition of solution space: after constructing a temporal net, each edge is associated to a set of one or more primitive qualitative temporal relations (called a compound temporal relation). Let N_i be the number of primitive relations associated to edge E_i. Let E_{max} be the number of edges in the temporal net. Then we define

$$\text{Solution Space Size} = \prod_{i=1}^{E_{max}} N_i \ .$$

8.3.2 Space Complexity of TIP

Space complexity: TIP 2.0 requires $O(N^2)$ memory locations to represent a fully connected temporal net. This can be easily deduced from the number of edges $E \sim N^2$.

Using state of the art solid state memory, the following net sizes kan be kept in fast primary memory (not on hard disks). One node consumes approximately 48 bits (PC version) or 64 bits (VAX version) to represent one qualitative and quantitative relation of a net (i. e. one interval / edge). For fully connected nets, this allows for a representation of nets up to 150 nodes on a conventional IBM-compatible PC, or approx. 800 nodes on a PC/AT with 15 MByte extended memory. On a 40 MByte VAX, more than 1100 nodes can be kept in main memory. By using slower secondary (virtual) memory and/or nets which are not fully connected, even much larger nets can be accomodated for.

Thus we conclude that space complexity of qualitative temporal logic is of no concern on state-of-the-art computer hardware.

8.3.3 Time Complexity and Consistency of TIP

Time complexity is an algorithmical issue. Allen's three-consistent constraint propagation scheme is used in TIP 2.0 in two steps: to construct a net (i.e., to represent all possible solutions in one consistent net), and to make it unique (in order to find one or more solutions). Both steps have a time complexity of $O(N^3)$: three-consistency means to iterate over all triangles (even repeatedly), until each triangle fulfills Allen's transitivity table. For there are $T \sim O(N^3)$ triangles and only a finite number of $s <= S = \text{const}$ iterations, until net stability is reached in step 1, and until net uniqueness is reached in step 2.

In scientific papers, discussion goes whether Allen's logic is "intractable" [literal meaning is "unmanageable"; intended meaning: of exponential growth] due to its time complexity. This question is strongly related to the question of consistency of Allen's logic (meaning if we can be safe from receiving a constructed network from step 1 which could have been further reduced by a better construction algorithm; or if we seemingly found a solution from step 2 which is three-consistent according to Allen, but which turns out to be inconsistent (and thus non-existing)). When we regard not only triangles, but rectangles, pentangles ... N-angles of edges, etc., full consistency of the net can be proven. The latter is obviously NP-complete.

Using TIP 2.0, we can provide some practical answers to these questions:

Step 1 (net construction) was measured for a benchmark suite of temporal nets (cf. Appendix 1). Net construction time was found to be always of the order of milliseconds for the nets of this benchmark suite.

Step 2 (finding one, a few, or all solutions) was measured for the same benchmark suite. Finding only one solution (presumably guided by some heuristics) is always of the order of milliseconds for nets of the size used in the suite. Finding n solutions (i. e., a few solutions) requires on the average n times the time of the first solution, provided the solutions are evenly distributed over solution space. In contrast, finding all solutions may take arbitrarily much time because of the combinatorial explosion of the number of solutions in solution space. However, depending on solution space size (and the deegree of connectedness of a not fully connected temporal net), and after establishing an appropriate time series of the relation between solution space size, connectedness and time, estimates can be made on the time needed to explore a particular solution space for all solutions. This estimate can then be fed back to further focus search by heuristics, until an acceptable run time and solution space size is found.

Consistency: it may be true that the question of consistency of any temporal net of step 1 or 2 is NP-complete). In the area of CIM however [this is a conjecture derived from practical observations, which remains to be proven theoretically yet], we are not concerned with the consistency of step 2, because here consistency is always granted to us: any qualitative solution found might also be a quantitative solution, because we are only interested in quantitative solutions (i. e. solutions that can be mapped upon the time line). But quantitative solutions mapped on the time line are always consistent, by definition. Any other qualitative solutions with no successful mapping on the time line (be it qualitatively consistent or only imaginary consistent) are rejected anyway. This works as long as inconsistent qualitative solutions are not produced to such an extent that TIP 2.0 spents its whole run time with finding and rejecting them. Practical observations indicate that TIP 2.0 rejects qualitative solutions but finds sufficient quantitative solutions, too. [An indicator is the ratio of the time spent on L-type solutions to the time spent on N-type solutions; cf. Appendix 1].

8.3.4 Hardware Acceleration and Heuristics

Imagine we had a net which TIP 2.0 would traverse finding all solutions in 1 hour. Assume, a hardware accelerator (a special-purpose coprocessor) would reduce this time by a factor of 1000. Now, all solutions would be available in 4 seconds. Assume that each edge in this net has (on the average) 3 primitive relations, which need to be made unique to find a solution. So if we added another 6 nodes, solution space would grow by a factor $3^6 = 1215$. Again, finding all solutions would take 1 hour !

Obviously, it is not possible to beat combinatorial explosion this way. Hardware acceleration is useful, but only when combined with heuristic pruning of solution space. Thus, the domain-heuristic module and the domain-to-solution space translator play an important role in Figure 1.

The efficient use of second-order constraints (meta constraints, if-then rules, demons) plays a another very important role in the pruning of solution space. Second-order constraints are furthermore *domain-dependent* knowledge (Figure 1).

9. User Interface Guidelines

9.1 TIP Main Loop

TIP is a fast temporal inference engine (constraint propagator). TIP is not intended for interfacing with a user; it is rather designed to be a functional module in a larger system. Thus, TIP's user interface is simple.

Note: 'situation' in the context of TIP is always a description of the qualitative and quantitative temporal relations under consideration, which is kept in an ASCII file. (Cf. [Jakob 1991a] for an introduction into the language which describes temporal situations).

TIP performs the following steps:

(1) Initialize TIP
(2) Query user for the current situation file, or possible exit (i. e. go to (8))
(3) Read and interpret situation file
(4) Display statistics on the situation file interpreted
(5) Perform propagation of qualitative and quantitative temporal constraints, according to the current situation, and the configuration found in the configuration file
(6) Display statistics on the propagation performed
(7) Go to step (2).
(8) End of TIP run.

These steps can be easily identified in the screen output of a typical TIP session:

```
TIP Version 2.0

(c) 1990, 1991 Philips GmbH Forschunglaboratorium Hamburg


Reading tables from TIP DATA FILE ...


Reading CONFIGURATION FILE ...


Enlarging transitivity table to SUM OF RELATIONS = 27 ...
... done !

Enlarging transitivity table to SUM OF RELATIONS = 29 ...
... done !

Testing commutativity of transitivity table ...
... passed !
```

Reading SOPO RULE FILE ...

+++ Underlying temporal logic has 13 simple relations.

Make a choice: 0 = SITUATION 1 = TEST 9 = EXIT: 0

Enter situation file name: Reading situation from file case1.sit

+++ There are 4 nodes and 6 edges.
+++ Logarithmic size (base 2) of solution space is 3.0
+++ Size of solution space is 8
+++ Setting up net took 0.170 s.: 0.028 s./edge 0.042 s./node.

Propagating constraints, please wait ...

+++ 6 L-solution(s) found !
+++ 6 N-solution(s) found !
+++ Percentage of L-solutions to solution space size is 75.00
+++ Percentage of N-solutions to solution space size is 75.00

+++ Run time was	1.040 s.
+++ One L-solution took	0.173 s.
+++ One candidate L-solution took	0.13000 s.
+++ One N-solution took	0.173 s.

Make a choice: 0 = SITUATION 1 = TEST 9 = EXIT: 9

End of TIP 2.0 run. Press RETURN to continue ...

In the near future, this purely line-oriented user interface will be encapsulated inside a graphical shell for Microsoft Windows 3.1.

9.2 TIP Configuration File

The following options can be set in the TIP configuration file (called 'TIP.CFG', to be found in the same directory as 'TIP.EXE'), influencing the run time behaviour of TIP (Software). An option is either a switch (in which case the boolean values FALSE or TRUE can be assigned to it, and it is called a logical predicate, denoted by '-P'), or an integer. If an option is not included in 'TIP.CFG', a default value is assumed.

FuelMax: allowed maximal no. of quantitative or qualitative solutions (depending on the value of QUANTITATIVEP) to be found.

TestSolutionP: determines if a solution found shall be tested for various consistency checks. A safety feature of TIP. Default is FALSE.

TraceP: if TRUE, a trace of the constraint propagation is sent to the screen. Verbose. Only useful on small examples. Default is FALSE.

CounterP: enables a file trace to file 'COUNT.DAT' of the constraint propagation performed. In contrast to TraceP, the file trace takes a rather compact form which reminds to a large digital counter. Default is FALSE.

QuantitativeP: TRUE if also quantitative constraint propagation shall take place.

StopP: Determines if TIP has to stop after each quantitative solution found and display the edge values. Verbose. Only useful for small situations. Defaults to FALSE.

DisplayP: determines if visual display of current "propagation depth" (one of 1 .. TemporalNet.EdgeNumber) is desired. Slows down ALLEN very much.

WriteSolutionsToFileP: determines if any quantitative solution found shall be written to file SOLUTIONFILE. Defaults to FALSE.

ProbabilisticP: determines if extended transitivity table (probabilistic CONSTRAINTS-P) shall be used. Otherwise Allen's original table is used. Using Allen's original table results in an average speed decrease of a factor of 3 to 4. Defaults to TRUE.

An example for a configuration file is the file 'TIP.DEF', which is the default configuration provided with the TIP sources. Note that one and only one option per line may appear:

```
"
QuantitativeSolutions     true
TestSolutions             false
Fuel                      0
Trace                     false
DisplayDepth              false
StopAfterSolution         false
WriteSolutions            false
CounterMode               false
"
```

10. Hardware And Software Requirements

TIP (Software) was originally written on a PC/AT compatible computer using Turbo Pascal 5.0, and conforming to the ISO Pascal standard. It thereby has a high level of portability. Thus, TIP (Software) releases 1.0 and 2.0 were ported to VAX Pascal 3.0 without difficulties. Current availability and versions:

- TIP 2.0 for PC-compatible computers running MS/DOS;
- TIP 2.0 for VAX workstations running VAX/VMS.

An advantage of the VAX-based version is the nearly unlimited number of temporal nodes due to the large linear address space (and virtual memory). The current PC release of TIP is limited to the available 640 - 700 kByte of RAM (extended memory currently not supported).

11. Prototype Implementation Experiences

11.1 Software Implementation Experiences

TIP was motivated by the obvious drawbacks of its LISP-based predecessor TIC I [Philips 1989]. TIC I was an excellent device for understanding the basics of temporal logic and of using temporal logic for planning tasks, but

- TIC I was too slow on scheduling problems with six or more jobs;
- TIC I had some deficiencies in expressing non-temporal constraints which are typical for scheduling tasks.

The design of TIC I was done having in mind to understand and experiment with temporal logic (as described by [Allen 1983] [Rit 1986]), and not so much to design for speed. The design of TIP (Software) started thus with some basic assumptions:

- choosing an imperative language as an implementation language rather than Lisp[2]. Motivations:
  - a typical Lisp system does not compile source code to an executable binary form, but only to an intermediate virtual machine representation, which is time-consuming to interpret (typically a factor of 3 to 10 slower than a genuine binary file).
  - a typical Lisp system uses symbols for denomination of enitities. Symbol maintenance however is comparitively slow: in an imperative language, integers are used to that end.
  - the Common Lisp language standard is so all-embracing that the average Common Lisp System implementor does neither design its implementation for speed, nor for full compliance to the standard. Often, (relatively) low-level functions (example: BOOLE-XOR et al. under VAX Lisp 2.2 and Genera 7.1) are not supported. These functions, however, are very useful for *speeding up* a system.
  - Lisp systems perform a certain amount of type-checking and garbage collection at run time, both hidden to the user (except by slowing down program execution). In imperative languages, the implementor is responsible herself or himself for type-checking, object creation and destruction, being able to withhold unnecessary actions.
- implementing sophisticated data structures in place of the known bottlenecks of TIP I and temporal logic:

---

[2] Still it is the author's opinion that there is nothing wrong with (Common) Lisp systems (interpreters, compilers and such), except for the poor level of implementation usually encountered. There is no reason why compiling a Lisp source file should not produce true binary code instead of virtual machine code (remark: it is true that EVAL makes things more difficult than in imperative languages). There is no reason for not keeping Lisp systems as simple as necessary (the Lisp dialect SCHEME is such an approach).

- the basic inference step of temporal logic (transitive closure) was speeded up using a probabilistic algorithm [Jakob 1990a] [Horowitz and Sahni 1981].

- the simple queue structure of Allen's ToAdd algorithm can be replaced by a recursive design of ToAdd, or by an improved queue design [Jakob 1990].

- use of integers for the denomination of entities (temporal relations, nodes, edges) in place of Lisp symbols allows for efficient "look up" data structures.

- use of bit vectors for the representation of temporal relations, instead of lists as in typical Lisp-based temporal reasoning systems (e. .g [Koomen 1989]).

These ideas were successfully incorporated in the design of TIP (Software). As a net effect, TIP (Software) outperforms TIC I by an average factor of 10 when run on the same hardware platform (confer Appendix 1, "TIP Performance Measurements" below).

Furthermore, the expressive power of qualitative temporal logic according to Allen is restricted to the temporal domain, erecting certain artificial barriers between temporal and any other knowledge. TIP (Software) performs a first step towards relaxing these barriers by allowing for simple arithmetical constraints, as well as second-order constraints (if-then rules, demons). Confer [Jakob 1991a] [Becker 1991] .

11.2 Hardware Implementation Experiences

In a second step, it was attempted to improve on TIP (Software) by designing a special hardware accelerator, called Temporal Inference Processor or "TIP (Hardware)". Until now, the following steps have been taken:

- the central chip "Temporal Inference Unit" for qualitative temporal reasoning was recognized, formally described [Jakob 1990b], designed and successfully implemented [Jakob 1991b];

- an ISA/EISA bus interface logic board was designed and implemented;

- design of quantitative temporal reasoning accelerator chip finished; implemented not yet started.

The Temporal Inference Unit (TIU) product description [Jakob 1991b] is summarizing the experiences gained when designing application-specific integrated circuits for temporal logic.

There is another important aspect to successful hardware design, however. During hardware design, software algorithms are deprived of any "fat", they are made as simple and fast as possible. This is true of the experiences gained with the Temporal Inference Unit (TIU) as well [Jakob 1991b]. Designing for speed, it turned out that qualitative temporal reasoning is performed fast when the following criteria hold:

- Any functions and routines transforming between different "coordinate systems" (edge-based, node-based) should be avoided during qualitative reasoning. Only "edges" (temporal relations) are used; the idea of a "node" (temporal interval) seems to be only useful for communication with human users, and during quantitative reasoning;

- There is no need for a stack or queue of changed edges in hardware design. However, if depth-first search shall be directed in any form, these data structures are desirable for re-arranging edges to-be-propagated according to the strategy chosen;

These (hardware) experiences should be fed back into future release of TIP (Software) then.

12. Software Module Structure

12.1 Modules

TIP consists of the software modules depicted in Figure 7.

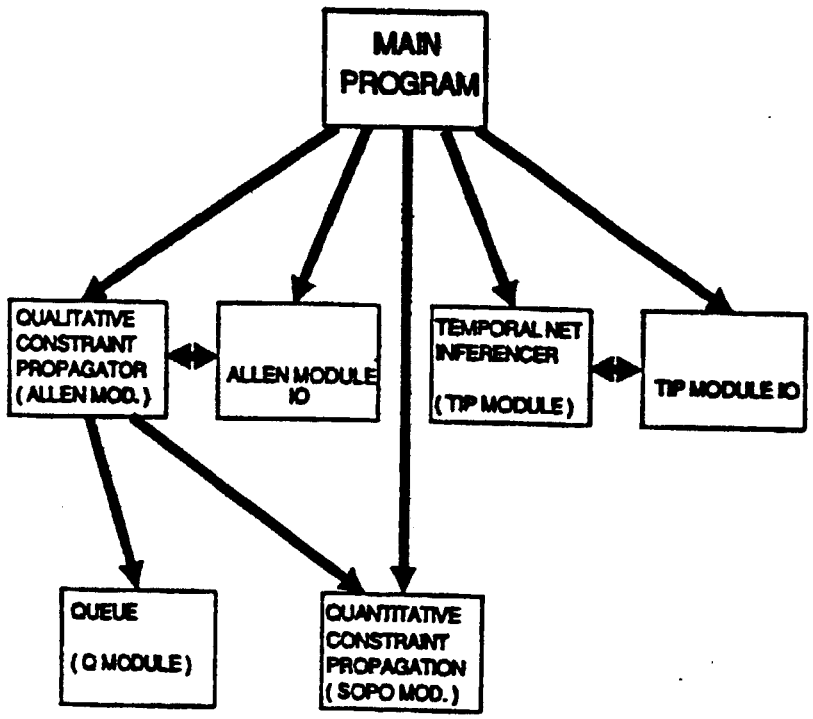


Figure 7: TIP (Software) Software Modules

Explanation of modules:

- ALLEN: concerned with the representation of a temporal net as data structures (its qualitative and some quantitative aspects). The constraint propagator. (Consists of the source files ALLECONS, ALLETYPE, ALLENVAR, ALLESUBR).

- ALLENIO: associated to the ALLEN module as an interface to the surrounding world. (Source file ALLENIO).

- TIP: this part of TIP is concerned with the (binary and word-oriented) internal representation of temporal relations. Transitivity amongst three connected edges (three temporal net edges which form a "triangle") can be computed from a fast transitivity table and its associated routines. The temporal inferencer. (Consisrs of the source files TIPCONS, TIPTYPE, TIPVAR, TIPSUBR).

- TIPIO: associated to the TIP module as an interface to the surrounding world. (Source file TIPIO).

- SOPO: representation of temporal intervals by their earliest start and finish, latest start and finish and minimal and maximal duration (so-called sopos). Interval arithmetic system. This representation allows for uncertainty in the quantitative data, which is minimized by constraint propagation amongst temporal intervals. (Consists of the source files SOPOCONS, SOPOTYPE, SOPOVAR and SOPOSUBR).

It is convenient to install TIP in a directory of its own. TIP consists of the following files:

- Primary source file: TIP.PAS. This file contains TIP's main program and includes all other (secondary) source files. Its compilation produces TIP.EXE.
- Secondary source files (all with extension .PAS): ALLECONS, ALLETYPE, ALLENVAR, ALLESUBR, ALLETEST, SOPOCONS, SOPOTYPE, SOPOVAR, SOPOSUBR, TIPCONST, TIPTYPE, TIPVAR, TIPSUBRO.
- One executable (binary) file: TIP.EXE. This file must be called to start TIP.
- Primary data files:
- TIP.DAT: contains all 'knowledge' on Allen's qualitative temporal logic. Is read by TIP.EXE upon initialisation. By modifying this file, different kinds of qualitative temporal logic (different from Allen's original logic) can be created and used.
- SOPORULES.DAT: contains all 'knowledge' on Rit's sopos (quantitative time logic). Is read by TIP upon initialisation. By modifying this file, different kinds of quantitative temporal logic (different from Rit'soriginal logic) can be created and used.
- TIP.CFG: TIP's configuration file, which can be used to configurate TIP according to your needs.
- TIPDOCU.ASC: this documentation (ASCII file).

Secondary data files:

- TIP13.DAT: needs to be copied onto TIP.DAT in order to invoke Allen's original, thirteen-valued qualitative temporal logic.

- TIP6.DAT: may be copied onto TIP.DAT, invoking a simpler and faster, six-valued temporal logic (which is a subset of Allen's). This logic does not possess the full descriptive power of Allen's logic.

- TIP3.DAT: may be copied onto TIP.DAT, invoking three-valued temporal logic which is known as the time-point logic of Villain and Kautz.

- SOPO13.DAT: needs to be copied onto SOPO.DAT in order to invoke a quantitative temporal interval logic (as described by Rit) which is compatible to Allen's thirteen-valued logic (cf. file TIP13.DAT).

- SOPO6.DAT: needs to be copied onto SOPO.DAT in order to invoke a quantitative temporal interval logic (as described by Rit) which is compatible to the six-valued logic described above (cf. file TIP6.DAT).

- SOPO3.DAT: needs to be copied onto SOPO.DAT in order to invoke a quantitative temporal time point logic (as described by Rit) which is compatible to Villain's three-valued logic (cf. file TIP3.DAT).

Command (batch) files:

- TO13.COM or TO13.BAT: copies TIP13.DAT and SOPO13.DAT onto TIP.DAT and SOPO.DAT.

- TO6.COM or TO6.BAT: copies TIP6.DAT and SOPO6.DAT onto TIP.DAT and SOPO.DAT.

- TO3.COM or TO3.BAT: copies TIP3.DAT and SOPO3.DAT onto TIP.DAT and SOPO.DAT.

During a TIP run, the following files may be created (but depending also on the switches set in TIP.CFG):

- COUNT.DAT: a trace file which documents in condensed form how TIP traversed the solution space of the last situation.

- TIP.SOL: file which contains all qualitative and quantitative solutions found during a TIP run, in textual form.

Furthermore it is useful to create a subdirectory containg all 'situation' files (all having an extension .SIT) provided. Situation files as provided contain example or benchmark files which describe input to TIP in a textual manner. The input of your particular application must be given to TIP in the same way . (A description of the input language used is given in [Jakob 1991a]).

12.2 TIP (Software) Main Loop

1 Initialize CONFIGURATION record
2 Interpret user configuration from CONFIGURATION file
3 Initialize Transitivity Table
4 Initialisize Relation Table
5 Read Transitivity Table from file TIP.DAT
6 Read inverses of simple relations from file TIP.DAT
7 Read symbols (strings) of primitive relations from file TIP.DAT
8 Compute all (compound) inverse relations from inverse relations in the relation table
9 Compute probabilistic (enlarged) Transitiviuty Table
10 Check commutativity of Transitivity Table
11 Initialize Sopo Rule Table
12 Read SOPO.DAT file, which contains currently valid sopo rules
13 Initialize interval arithmetics (so-called Interval System)
14 Initialize Allen's queue 'ToDo'
15 Initialize all sopos (create them as a dynamical data structure)
16 WHILE user wishes to continue

17 clear all sopos (assign zero / infinity to MIN and MAX fields)
18 Query name of SITUATION file from user
19 Interpret SITUATION from specified file. Creates the current temporal net, and the current sopos are read.
20 Estimate (statical) solution space of current situation
21 Display estimate and interpretation time to user.
22 Make a 'snapshot' of the current temporal net and sopos, before making the temporal net unique. Store snapshot in memory.
23 Make current temporal net unique. (A recursive routine which stops at each qualitative and quantitative solution, writing them to SOLUTIONS file if required by CONFIGURATION record. Backtracks for finding further solutions.)
24 Display number of solutions found and run time to user.

25 End of main

13. Software Modules Interface Description

13.1 Documentation of the ALLEN Module (Files ALLECONS, ALLETYPE, ALLEVAR, ALLENIO, ALLESUBR)

Module Summary:

The ALLEN module is the (historically) second-oldest part of TIP. It centres around a representation of qualitative temporal nets as described by Allen [Allen 1983]. The center of gravity was laid upon a time-efficient representation of the net. (Space efficiency is not considered to be an existing problem on contemporary computer hardware).

Objects (Module Variables):

The important objects of the ALLEN module are

- TEMPORALNET: a global record containing the fields ...

- NODENUMBER, NODENUMBERPLUS1, EDGENUMBER: the number of edges or nodes (or nodes plus 1) of the current temporal net.

- SOLUTIONS, QUANTITATIVESOLUTIONS: the absolute number of solutions found up to now, differentiated according to purely qualitative (SOLUTIONS), and qualitative and quantitative solutions (QUANTITATIVESOLUTIONS). Note that always SOLUTIONS >= QUANTITATIVESOLUTIONS.

- the NET field: a vector of length EDGENUMBER. Each vector entry contains the subfields EDGE (the currently valid compound relation of the indexed edge), SNAPSHOT (a one compound relation memory which serves to remember the compound relation of EDGE after making the temporal net nonambigous (after qualitative propagation), but prior to restricting the net further by quantitative values. SIMPLERELATIONMARKER is a "simple relation" (one of [1 .. NOOFSIMPLERELATIONS]); it serves as a TIP internal marker during the process of making edges nonambigous. The FROM and ONTO fields contain the intervals (nodes) which are connected to this edge (without being changed during the program run). The PROBE fields serves to enable or disable tracing the constraint propagation throughout the net.

- NOINFORELATION : Allen called the union of all primitive temporal relations 'no info'. The distinctive binary representation of this compound temporal relation is kept in this variable. Its value always is 2 NOOFSIMPLERELATIONS - 1.

- SOLUTIONSPACESIZE : each temporal net can be assigned a "solution space size" S. This size equals the cardinality of the set of possible edge instantiations by primitive relations, which is depending on the number of allowed primitive relations per edge. Let A_i be the number of allowed primitive relations for edge i. Then we receive

$$S = \prod_{i=0}^{\text{TEMPORALNET.EDGENUMBER}} A_i$$

For practical reasons, SOLUTIONSPACESIZE is the natural logarithm of S as defined, to avoid floating point overflow.

- SITUATIONFILENAME : This file is the current temporal situation file, as specified by the user in the configuration file.

- CONFIGURATION : TIP requires the setting of some switches and the knowledge of some file names before it runs. This information is kept in the CONFIGURATION record. Record fields are:

- FUELP, FUELMAX: switch to indicate whether only a limited number of solutions shall be found or not. If true, FUELMAX will contain the upper limit of solutions to be found.

- COUNTERP, THECOUNTER: switch to indicate whether the course of constraint propagation throughout the net shall be logged to a file (the COUNTERFILE). Since all edges are considered to be in a lexicographical ordering, and are written to this file according to this ordering, the user gains the impression of a large n-ary counter unfolding when consulting COUNTERFILE. The THECOUNTER field is associated to COUNTERP: it is a kind of time stamp attached to each counter reading written to COUNTERFILE. [The (always increasing) time "tick" is the number of calls to a TIP-internal function called GETNEXTSIMPLERELATION].

- TESTSOLUTIONP : a switch indicating whether a quantitative solution found shall be checked for consistency with the situation and temporal net initially provided. Three checks are applied: solution versus qualitative situation, solution versus quantitative solution, qualitative versus quantitative solution.

- TRACEP: a switch indicating whether tracing shall be on or off. "Tracing" refers to the constraint propagation between sopos (set of possible occurences), i. e. the quantitative constraint propagation. On the terminal, the ongoing process of constraint propagation is monitored alphanumerically.

- QUANTITATIVEP : if false, only quantitative solutions will be searched and found.

- STOPP: true if the program shall stop after each solution found, and display the solution.

- DISPLAYP: true if visual display of current "propagation depth" [one of 1 .. TEMPORALNET.EDGENUMBER] is desired. Slows down TIP very much.

- WRITESOLUTIONSTOFILEP: switch indicating whether the solutions found shall be written to the SOLUTIONSFILE, or not.

13.2 Documentation of the TIP Module (Files TIPCONST, TIPTYPE, TIPVAR, TIPIO, TIPSUBRO)

Module Summary:

The TIP module is the (historically) oldest part of TIP. It centres around Allen's transitivity table [Allen 1983] and an extended, 'probabilistic' extension thereof [Jakob 1990a]. A transitivity table is read from file and kept in RAM. However, the underlying temporal logic is not restricted to the thirteen-valued temporal logic as introduced by Allen; arbitrary temporal logics can be defined and represented [Jakob 1990b].

Objects (Module Variables):

The important objects of the TIP modules are

- TIPDATAFILE:

- (1) contains the number of primitive compound relations, in case of Allen's logic this number is 13. This number is kept in a global variable NOOFRELATIONS. An empty line must follow this entry in TIPDATAFILE.

- (2) contains Allen's transitivity table (or any other transitivity table specific for an arbitrary, user-defined temporal logic). Each table entry occupies a line, where the first two numbers denote the first and second primitive relation (row and column index, or "address" of the entry) respectively. The remaining numbers of each line denote the primitive relations that occupy this entry. These primitive relations are coded by numbers, they are always member of [1, NOOFRELATIONS] . The 'meaning' of these code numbers is defined later in TIPDATAFILE. An empty line must follow this entry in TIPDATAFILE.

- (3) contains the inverses of each primitive relation. Each line contains a primitive relation, then its inverse. An empty line must follow this entry in TIPDATAFILE.

- (4) contains strings or 'symbols' that identify the integer-coded primitive relations. One line contains a primitive relation number (code), then its associated string.

- TRANSITIVITYTABLEFILE: this file contains the extended transitivity table, i. e. the extended table as internally represented in TIP. Each line contains line and row index (first and second primitive relation), and its associated COMPOUND relation IN INTERNAL REPRESENTATION form; i. e. as an integer value [DONTCARERELATION .. TOTALRELATIONS] .

- NOOFSIMPLERELATIONS: the number of simple relations as specified in TIP.DAT (Tip Data File).

- TRANSITIVITYTABLE:

- MAXINDEX: the total number of transitive relations kept in the transitivity table AFTER enlarging the transitivity table. Usually larger than NOOFSIMPLERELATIONS.

- TABLE: the transitivity table, a two-dimensional matrix where each entry is the resulting temporal compound relation (gamma edge). Alpha and beta edge provide the respective row and column index. Note: alpha and beta edge compound relations must be mapped to their associated primitive relations (one of 1 .. MAXINDEX) before using them as row or column index.

- RELATIONTABLE:

- SUMOFRELATIONS: how many of the records provided in the empty relation table are actually used by the current temporal logic ? SUMOFRELATIONS contains this number.

- TABLE: contains SUMOFRELATIONS pointers to records. Records are either addressed by (1) their compound relation (a kind of associative memory principle) under consideration, or by (2) their index in RELATIONTABLE. Each record consists of:

- INDEX: when using a compound relation for addressing memory, INDEX contains the index under which this relation is found in RELATIONTABLE. Needs a second access of RELATIONTABLE using this index.

- INVERSERELATION: the index of the associated inverse relation.

- RELATIONBYINDEX: when using a table index for addressing memory, RELATIONBYINDEX contains the relation which is associated to this index.

- SIMPLERELATIONINVERSE: a vector (one-dimensional table) where a primitive relation, used as a table index, can find its associated inverse primitive relation.

- SYMBOLTABLE: a vector (one-dimensional table) where a primitive relation (represented as an integer), used as a table index, can find its associated inverse primitive relation, represented as a character string.

13.3 Documentation of the SOPO Module (Files SOPOCONS, SOPOTYPE, SOPOVAR, SOPOSUBR)

Module Summary:

The SOPO module has its name from the 'set of possible occurences' (sopo) of [Rit 1986]. A sopo is a 6-ary relation describing time intervals, consisting of earliest and latest start, finish and duration of the interval. In a sense, sopos are a quantitative extension of Allen's temporal logic [Allen 1983]. In contrast to the sopos described by [Rit 1986], the sopos implemented here describe only time intervals which do not contain any "time holes".

In a first step, the SOPO module is initialized by reading a SOPORULEFILE. The SOPORULEFILE contains a verbal description of certain properties of Allen's primitive temporal relations when applied to a quantitative time domain. This description is read, and an internal representation of it is kept in the SOPO module (so-called "sopo rules") in the SOPORULETABLE global variable. Whenever constraint propagation of Allen's temporal relations shall propagate quantitative temporal information between intervals, SOPORULETABLE is consulted for applicable rules. If rules are found in SOPORULETABLE, a small interpreter applies them to the quantitative information of the current temporal net, thereby constraining quantitative time intervals. The quantitative information on the current temporal net is kept in a global variable INTERVALS.

An advantage of describing sopo rules verbally is that they can be accomodated to arbitrary temporal logics, without the need to change any source code statement.

Objects (Module Constants and Variables):

The objects (variables) of the SOPO module are:

- UNDEFINEDINTERVAL : a constant (in PASCAL: variable) representing the (outstanding) undefined interval.

- INTERVALS : the quantitative side of the current temporal net. Each node gets two associated sopos (called SOPO and SNAPSHOT fields). The SOPO field holds the current value, whereas the SNAPSHOT field is used to memorize the value of this SOPO as supplied by the SITUATIONFILE (before propagating any constraints on sopos).

- SOPORULETABLE : a table containing the internal representation of the sopo rules, as verbally read in from the SOPORULEFILE. Fields:

- MINIMAX : each rules applies to the 'minimum' or 'maximum' of any two intervals (that are connected by an edge) under consideration. Minima are the earliest start, earliest finish and minimal duration of an interval; maxima are the lastest start, latest finish and maximal duration of an interval.

- OPERATOR : each rule uses one operator which is used to further constrain the two intervals connected by one edge. Currently, the operators known are <, <=, =, >=, >.

- TYPEOFINTERVALA, TYPEOFINTERVALB : each rule applies to a certain part of a sopo only. The involved part of the first interval is named TYPEOFINTERVALA, the involved part of the second interval is named TYPEOFINTERVALB. Values for TYPEOFINTERVALA/B are either start, finish or duration of the sopo involved.

- NEXTRULE : pointer to the next rule valid for this particular primitive temporal relation. All such rules (for one particular primitive relation) are kept in a linear list.

- SOPORULEFILE : the file containing the verbal description of the quantitative constraints associated with particular primitive temporal relations. One line contains one rule. The line format is "SOPORULE <PRIMITIVE-TEMPORAL-RELATION> <MINIMAX> <OPERATOR> <TYPE-OF-INTERVAL-A> <TYPE-OF-INTERVAL-B>". Lines need not be in any particular order.

The important objects (constants) of the SOPO module are:

- ZERO: the point of zero of the SOPO time interval system. By default, ZERO = 0.

- PLUSINFINITY: infinity for the SOPO time interval system. All quantitative time intervals use time points from the closed interval [ZERO, PLUSINFINITY]. By default, PLUSINFINITY = MAXINT div 2, to allow dyadic integer addition.

- TIMERESOLUTION : minimal time distance between two points in time. Used throughout constraint propagation, as SOPO uses an integer based arithmetic. Typically 1.

- SMALLESTDURATION: minimal time distance between start and finish of intervals; typically 1 (may also be 0).

13.4 TIP Intermodule Communication (Import and Export of Routines)

ALLENIO exports ...

AllenError
ConvertStringToBoolean
ConvertStringToCardinal
FindToken
GetInternalRunTime
LookupRelationSymbol
MakeDialog
UpcaseLine

ALLENSUBR exports ...

EstimateSolutionSpace
GetMaxRelation
InitConfiguration
InterpretFile
MakeSnapshot
MakeTemporalNet
MakeUnique
PutValueBetweenNodes
RestrictValueBetweenNodes

ALLETEST exports ...

TestAllen
TestSuite

SOPOSUBR exports ...

AdjustSopo
ApplySopoRules
InitIntervalSystem
InitAllSopos
ClearAllSopos
MakeSopoSnapshot
InitSopoRuleTable
RestoreSopos
LearnSopoRule

TIPIO exports ...

ReadSymbols
TipError

TIPSUBRO exports ...

ComputeAllInverseRelations
ComputeRelationOr
ConvertSimpleRelationToRelation
DisplayRelation
GetInverseRelation
GetTransitiveRelation
InitRelationTable
InitTransitivityTable
ReadInverses
ReadTransitivityTable
TestCommutativityOfTransitivityTable
TestForSimpleRelationInRelation
EnlargeTransitivityTable
WriteExtendedTransitivityTable

14. Training Manual

Being a prototype, there is no Training Manual available for TIP. However, a number of well-documented applications of temporal logic to planning and scheduling problems is available, including the problem formulation in the TIC I and TIP (Software) languages describing temporal situations. To name a few:

- so-called "Huber benchmarks" [Huber 1990] for testing qualitative and quantitative temporal logic on planning and scheduling problems (to be demonstarted in Appendix 1);
- AIPLANNER application;
- CPM application (both [Jakob 1991a]).

A description of the command-oriented TIP (Software) user interface is to be found in Chapter 9 (User Interface Guidelines).

15. Technical Summary

TIP (possible interpretation: "Temporal Inference in a Procedural language") attempts to provide a fast inference engine for temporal logic. It departs from Lisp's symbolic representation of time relations and rather uses bit vectors to that end. The intended minimal speed increase of one order of magnitude over a "conventional" Lisp implementation was already fulfilled by the first release, version 1.0.

Outstanding features of version TIP 2.0 are:

- order of magnitude speed increase relative to Common Lisp version on the same hardware platform.
- arbitrary choice of temporal logic. Three common logics are included: (1) Allen's well-known thirteen-valued, qualitative temporal interval logic, (2) six-valued subset of Allen's logic for another 100 % speed increase, (3) Villain's three-valued timepoint logic. Other types of temporal logic may be defined by the user in simple tabular form.
- quantitative extension including uncertainty, based on the concept of Rit's "sopo" time intervals. The arithmetic rules on sopos support the three different temporal logics included. The user may define his own sopo rule set as a file, to support a temporal logic of his own.
- a trace facility for qualitative reasoning, enabling the user to answer the why and how of the course of inferencing taken by the constraint propagator.
- a built-in probabilistic transitivity table, which takes many of the credits for the speed increase.
- priority control amongst edges of the temporal network (which edges shall change their assigned values more frequently under inferencing, and which edges shall keep their values as long as possible).
- tabular description of the properties of qualitative and quantitative temporal logic (compared to conventional implementations which use "hard-wired" logics which form part of the program source and are hard to modify).
- a queueless inference engine (further simplification of Allen's basic algorithm).
- "connectivity" measure of the network reports about the average number of edges connected to each edge.

TIP was fully written in ISO Pascal . Although its structure is similar to assembler programming, in that it provides a fast, low level temporal inference engine, it is easily portable from machine to machine. A few machine-dependant substitutions of Pascal routines serve as a further means to speed up TIP. TIP needs to be integrated into a larger system for temporal logic consisting of: (1) a domain-dependant expert system, (2) another object-oriented expert system that is able to translate domain-specific situations to domain-independent time relations, (3) a graphical user interface which must be added for a working system. Since TIP can be easily linked to or called from other programs (even written in other programming languages), integration should pose no diffuculties.

Inference engine for qualitative and quantitative interval logic. Based on Allen's qualitative temporal interval logic [Allen 1983], and on a subset of Rit's quantitative extension [Rit 1986] to Allen's logic.

Contrary to previously existing implementations of Allen's logic, temporal relations are represented as bit vectors rather than lists of symbols [Jakob 1990b]. The transitivity table is probabilistically extended to increase the hit rate [Jakob 1990a].

Written in ISO Pascal with proprietary extensions for file and string handling on VAXes (VaAXPascal 3.0) and IBM PC compatibles (Turbo Pascal 5.0).

Approximately 150 kBytes of source code, 64 kBytes of executable binary code (PC version).

64 kByte data space on PCs; unlimited (limited by virtual memory) data space on VAXes. Most of data space can be used to represent a temporal net. Default temporal net sizes are currently: 39 nodes (PC) / 499 nodes (VAX).One node equals one time interval. Extendable by simple recompilation to 150 nodes (640 kByte PC) or 800 nodes (16 MByte PC), 1100 nodes (40 MByte VAX) when using primary memory. Much larger temporal nets can be implemented by using virtual memory schemes and/or not fully connected temporal nets.

Space complexity of TIP 2.0: $O(N^2)$. N: number of nodes and temporal intervals of the problem under consideration.

Time complexity of TIP 2.0: $O(N^3)$.

Constraint propagation algorithm used: Allen's 'ToDo' algorithm, refined by a probabilistic transitivity table.

Check for full consistency of constructed net / minimal spanning solution constructed: No; but confer statement on consistency of quantitative solutions found.

Consistency of qualitative solutions found: three-consistency.

Consistency of the quantitative solutions found: full consistency (qualitative and quantitative).

Additional remedies against combinatorial explosion: heuristic pruning of solution space, hardware acceleration.

16. References

[Allen 1983] J. F. Allen, Maintaining Knowledge about Temporal Intervals; in. Comm. ACM, pp. 832 - 843.

[Becker 1988] S. Becker, Konzeption und Implementation einer temporalen Inferenzkomponente: Anwednung auf Planungsprobleme in der Fließproduktion, Diplomarbeit, Fachbereich Informatik, Universität Hamburg.

[Becker 1991] S. Becker, Qualitative Representation of Scheduling Problems; in: ESPRIT 2434 Consortium / Philips GmbH [ed.], ESPRIT 2434 30 Months Report, Task 2.5, Hamburg.

[Bünz 1989] D. Bünz, Horizontal Integration of CIM Modules: The Control Strategy; in: ESPRIT 932 Consortium / Philips GmbH [ed.], ESPRIT 932 36 Months Report, Task 2.3.3.

[Colinot and Lepape 1987] A. Collinot, C. Lepape, Controlling Constraint Propagation; in: Proc. 10th Int. Joint Conf. AI, Milan, pp. 1032 - 1034.

[Elleby et al. 1988] P. Elleby et al., Reactive Constraint-based Job-Shop Scheduling; in: M. D. Oliff [ed.], Expert Systems and Intelligent Manufacturing, pp. 1 - 10, Elsevier Science Publishers: Amsterdam.

[Fox 1983] M. S. Fox, Constraint-directed Search: A Case Study of Job-Shop Scheduling, PhD Thesis, Carnegie Mellon Univesity, Pittsburgh, Pennsylvania, USA.

[Haralick and Elliot 1980] R. M. Haralick, G. L. Elliot, Increasing Search Efficiency for Constraint Satisfaction Problems; in: Artificial Intelligence, pp. 263 - 313.

[Horowitz and Sahni 1981] E. Horowitz, S. Sahni, Algorithmen, Springer Verlag: Heidelberg. (Originally published in English).

[Huber 1990] A. Huber, Dynamic Scheduling under Different Production Strategies; in: ESPRIT 2434 Consortium / Philips GmbH [ed.], ESPRIT 2434 18 Months Report, Task 2.1, Hamburg.

[Jakob 1990a] J. - I. Jakob, Evaluation of Allen's Qualitative Temporal Inference Algorithm for Later Hardware Implementation; in: ESPRIT 2434 Consortium / Philips GmbH [ed.], ESPRIT 2434 12 Months Report, Task 2.5, Hamburg.

[Jakob 1990b] J. - I. Jakob, A Temporal Inference Accelerator Unit; in: ESPRIT 2434 Consortium / Philips GmbH [ed.], ESPRIT 2434 18 Months Report, Task 2.5, Hamburg.

[Jakob 1991a] J. - I. Jakob, TIP: An Intermediate Language for the Description of Temporal Situations; in: ESPRIT 2434 Consortium / Philips GmbH [ed.], ESPRIT 2434 24 Months Report, Task 2.5, Hamburg.

[Jakob 1991b] J. - I. Jakob, Temporal Inference Accelerator Board: Progress Report; in: ESPRIT 2434 Consortium / Philips GmbH [ed.], ESPRIT 2434 30 Months Report, Task 2.5, Hamburg.

[Jakob 1991c] J. - I. Jakob, Temporal Inference Unit -- Implementing An AI Coprocessor; in: ESPRIT 2434 Consortium / Philips GmbH [ed.], ESPRIT 2434 24 Months Report, Task 2.5, Hamburg.

[Jakob and Isenberg 1991] J. - I. Jakob, R. Isenberg, AIPLANNER Factory Layout Editor (AIPLANNER/FALO); in: ESPRIT 2434 Consortium / Philips GmbH [ed.], ESPRIT 2434 Final Report, Product Descriptions (Part 1), Hamburg.

[Koomen 1989] J. Koome, The TIMELOGIC Temporal Reasoning System, Technical report 231, University of Rochester, USA.

[Moder et al. 1983] J. J. Moder et al., Project Management with CPM, PERT and Precedence Diagramming, van Nostrand Reinhold: New York.

[Montanari and Rossi 1988] U. Montanari, F. Rossi, Fundamental Properties of Networks of Constraints: a New Formulation; in: Proc. Europ. Conf. AI, Munich, pp. 188 - 190.

[Nadel 1988] B. Nadel, Tree Search and Arc Consistency in Constraint Satisfaction Problems; in: Kanal and Kumar [ed.], Search in AI, pp. 287 - 342, Springer: New York.
[Parr 1991] R. Parr, Design of the Productio Module SOCO for Integration of Corrective Maintenance Requests; in: ESPRIT 2434 Consortium / Philips GmbH [ed.], ESPRIT 2434 30 Months Report, Task 2.3, Hamburg.
[Rit 1986] J.-F. Rit, Propagating Temporal Constraints for Scheduling; in: Proc. 5th AAAI Conf., Philadelphia, USA.
[Sacerdoti 1977] E. D. Sacerdoti, A Structure for Plans and Behaviour, Elsevier North-Holland: New York.
[Stefik 1981] M. Stefik, Planning with Constraints (MOLGEN # I, II); in: Artificial Intelligence, pp. 111 - 169.
[Swain and Cooper 1988] M. J. Swain, P. R. cooper, Parallel Hardware for Constraint Satisfaction; in: Proc. 7th Conf. of AAAI, pp. 682 - 686.
[Villain and Kautz 1986] M. Villain, H. Kautz, Constraint Propagating Algorithms for Temporal Reasoning; in: Proc. 5th AAAI Conf., Philadelphia, USA, pp. 377 - 382.
[van Hentenryck 1989] P. van Hentenryck, Constraint Satisfaction in Logic Programming, MIT Press: Cambridge, Massachusetts/USA.

Appendix 1:

TIP (Software) Performance Measurements

The "Huber Benchmarks"

The following benchmarks for temporal reasoning have been compiled by Dr. A. Huber in the beginning of ESPRIT 2434 [Huber 1990]. They contain simple planning and scheduling problems, described as temporal "situations". The benchmarks have been ported from TIC I's "input language" (not a real input language, but typically a call sequence of Lisp functions) [Philips 1989] to the TIP language [Jakob 1991a]. *Note:* The expressiveness of the TIC I input language is a subset of the expressiveness of the TIP language, so porting did not pose any serious difficulties.

Description of Benchmarks

"Testcase-1" (file CASE1): 3 order before-after chain (4 intervals).

"Testcase-1 with Additional Constraints" (CASE 1 E-XTENDED): 3 order before-after chain with additional 3 constraints on one interval (7 intervals).

"Testcase-2" (file CASE2): 6 order before-after chain (7 intervals).

"Testcase-2 with Additional Constraints" (CASE 2 E-XTENDED): 6 order before-after chain with additional 6 constraints on two intervals (13 intervals).

"Testcase-3" (file CASE3): 9 order before-after chain (10 intervals).

"Testcase-3 with Additional Constraints" (file CASE 3 E-XTENDED): 9 order before-after chain with additional 9 constraints on three intervals (19 intervals).

"Testcase-4" (file CASE4) demonstrates a way to synchronise two coupled flowlines in a just-in-time manner (12 intervals).

"Testcase-5" (file CASE5) is the well-known two machines problem (7 intervals).

Test Strategy

Each benchmark is tested for

- its setup time;
- its total time for finding ALL solutions;
- the time for finding the first solution;
- the number of qualitative (L-) solutions found;
- the number of quantitative (N-) solutions found.

All tests are performed in a way to allow for maximum speed (e. g., no protocol files are produced, etc.). Some tests allow for the use of subsets of Allen's interval logic (depending on their usage of primitive relations), but this fact has not been taken advantage of. A comparison between TIP (Software) and TIC I results are given, where reasonable. On some benchmarks, TIC I seemed to consume an infinite amount of time and was interrupted after one day, without producing a definite result. TIP (Hardware) has not been benchmarked, since its design has not yet fully been implemented[3].

Table 1: Run Time Measurements of "Huber Benchmarks" (Total Time and Setup Time)

	Solution Space Size	Finding All Solutions on Symbolics 364X [s]	Finding All Solutions on VAXStation 3500 [s]	Finding All Solutions on VAXStation 3500 [s]	Finding All Solutions on COMPAQ 486/25 [s]
Inference Engine Used		TIC I	TIC I	TIP 1.0[4]	TIP 1.0
CASE 1	$2^3 = 8$	3	3.09	0.31	0.21
CASE 1 E	$2^6 = 64$	5	11.11	1.52	1.16
CASE 2	$2^{15} = 32{,}768$	550	1458.10	137.1	81.79
CASE 2 E	2^{21}	3800		4496.2	2846.9
CASE 3	2^{36}			168,979.9	
CASE 3 E	$> 2^{36}$				
CASE 4	$2^{156.3}$			114.8	0.33
CASE 5	$2^{33.8}$	1750		2539.5	1622.7

Table 2: Run Time Measurements of "Huber Benchmarks" (Time Used Finding First Good Solution)

	First Good Solution on Symbolics 364X after [s]	First Good Solution on VAXStation 3500 after [s]	First Good Solution on VAXStation 3500 after [s]	First Good Solution on COMPAQ 486/25 after [s]
Inference Engine Used	TIC I	TIC I	TIP 1.0	TIP 1.0
CASE 1	2	1.33	0.088	0.077
CASE 1 E	1.5	3.20	0.083	0.121
CASE 2	6	18.08	0.290	0.174
CASE 2 E	16	24.07	1.571	1.072
CASE 3	19	83.90	0.646	
CASE 3 E	220	225.95		
CASE 4	180	> 1 day		
CASE 5	10	> 1 day	0.849	0.473

[3] Very preliminary timing results on TIP (Hardware) have been reported in [Jakob 1991c].

[4] Due to lack of time, only the measurements for TIP 1.0 are available. TIP 2.0 generally runs considerably faster than TIP 1.0 due to simplified data structures.

Interpretation of Tables 1 and 2: Of particular interest are the measurements performed on the VAXStation 3500, because they allow for a direct comparison between TIC I and TIP 1.0, and indirectly then of TIC I and TIP 1.0 in general. Some measurements have not been taken on all machines; in this case they are left blank. Other measurements do not terminate ("> 1 day"), which presumably is a software bug of the underlying inference engine. It turns out that TIP 1.0 achieves an average speedup factor of 10 or more over TIC I on the same hardware. Test cases 1, 2 and 3 form a series of simple scheduling tasks. As expected, run time for finding all solutions grows exponentially over the number of jobs scheduled. Here it is advisable to use if-then rules and other non-temporary constraints to prune solution space by the application of domain-specific knowledge.

Figures 8 to 14 depict the above test cases in qualitative temporal net representation (source: [Huber 1990]).

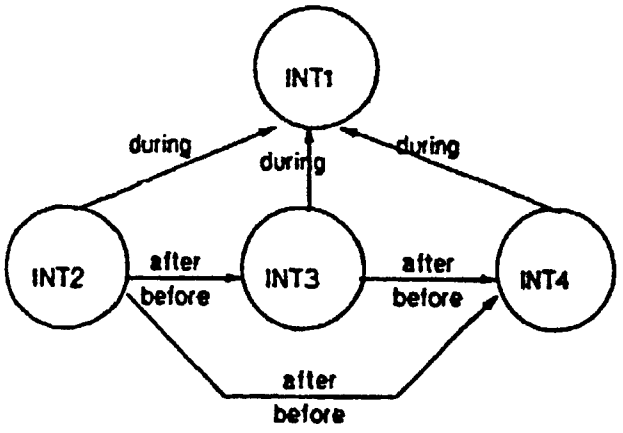


Figure 8: Test Case 1 (Three Order Schedule)

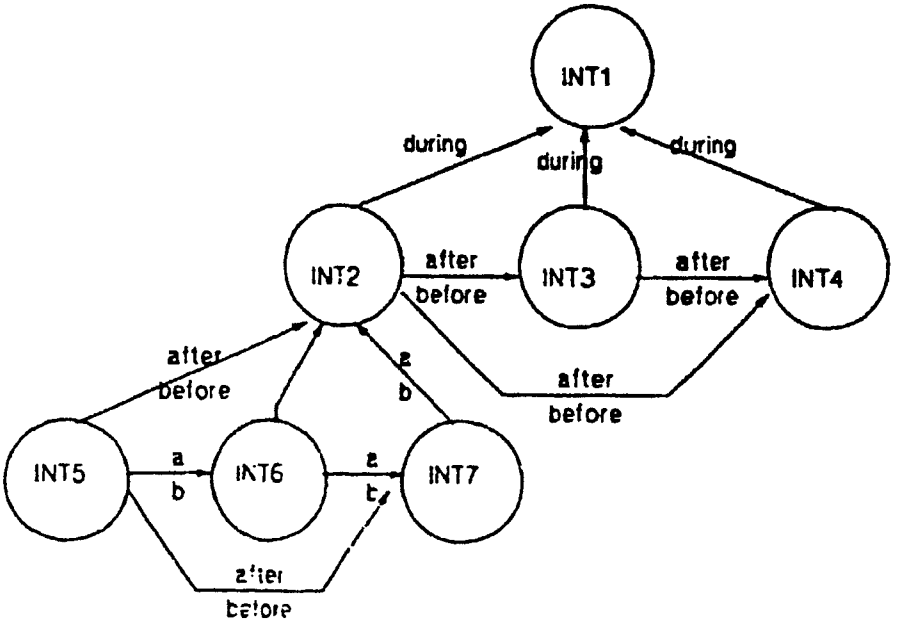


Figure 9: Test Case 1 Extended (Three Order Schedule with Additional Constraints)

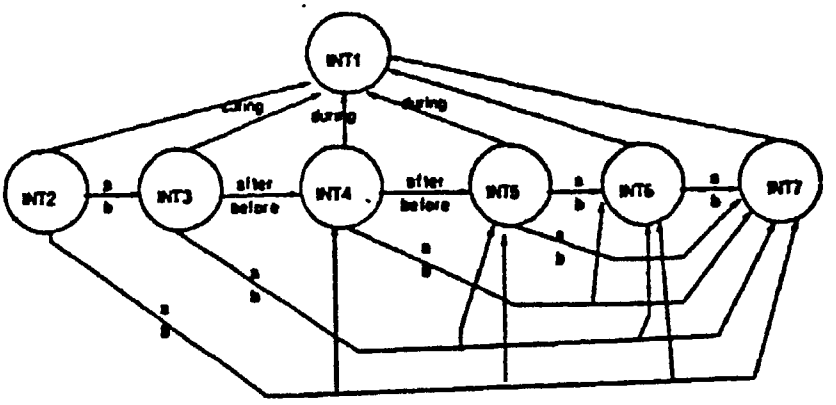


Figure 10: Test Case 2 (Six Order Schedule)

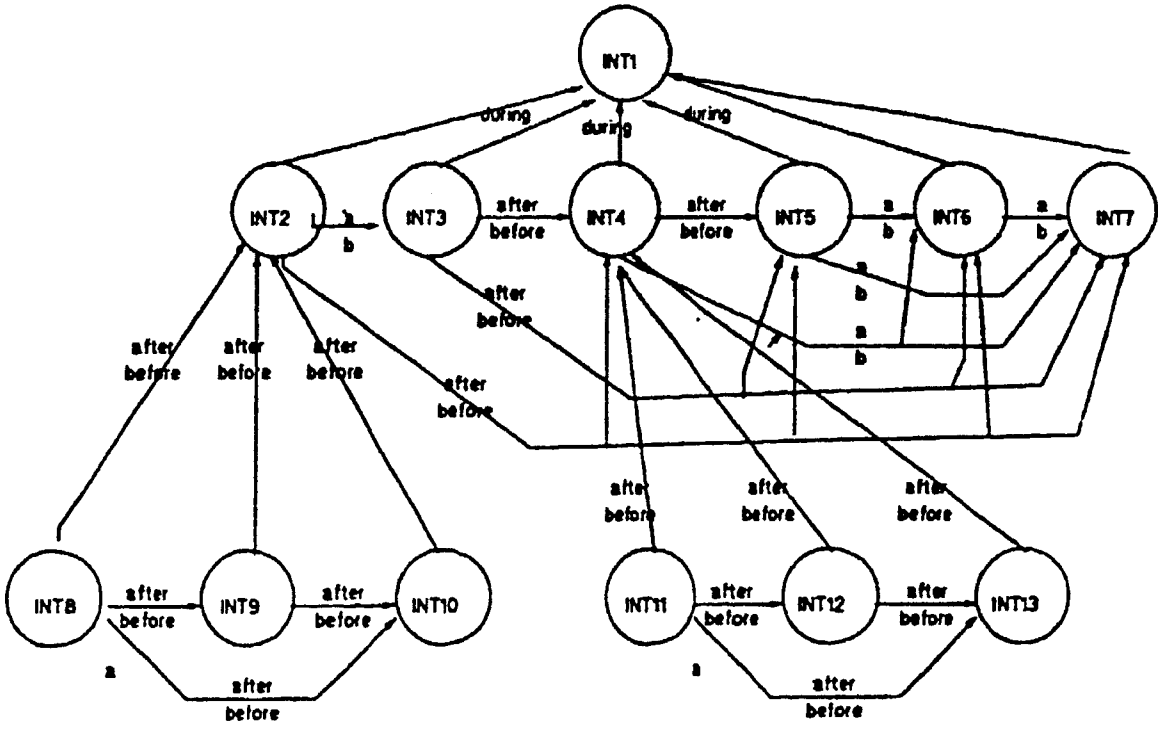


Figure 11: Test Case 2 Extended (Six Order Schedule with Additional Constraints)

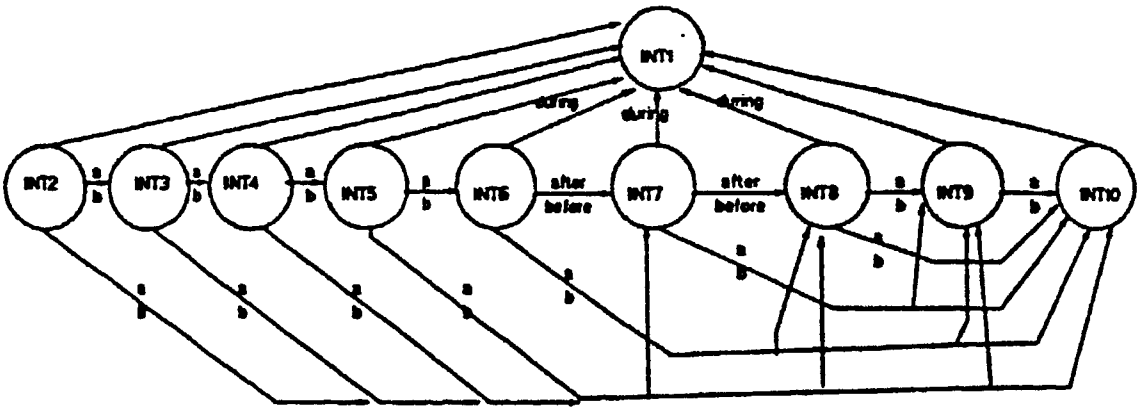


Figure 12: Test Case 3 (Nine Order Schedule)

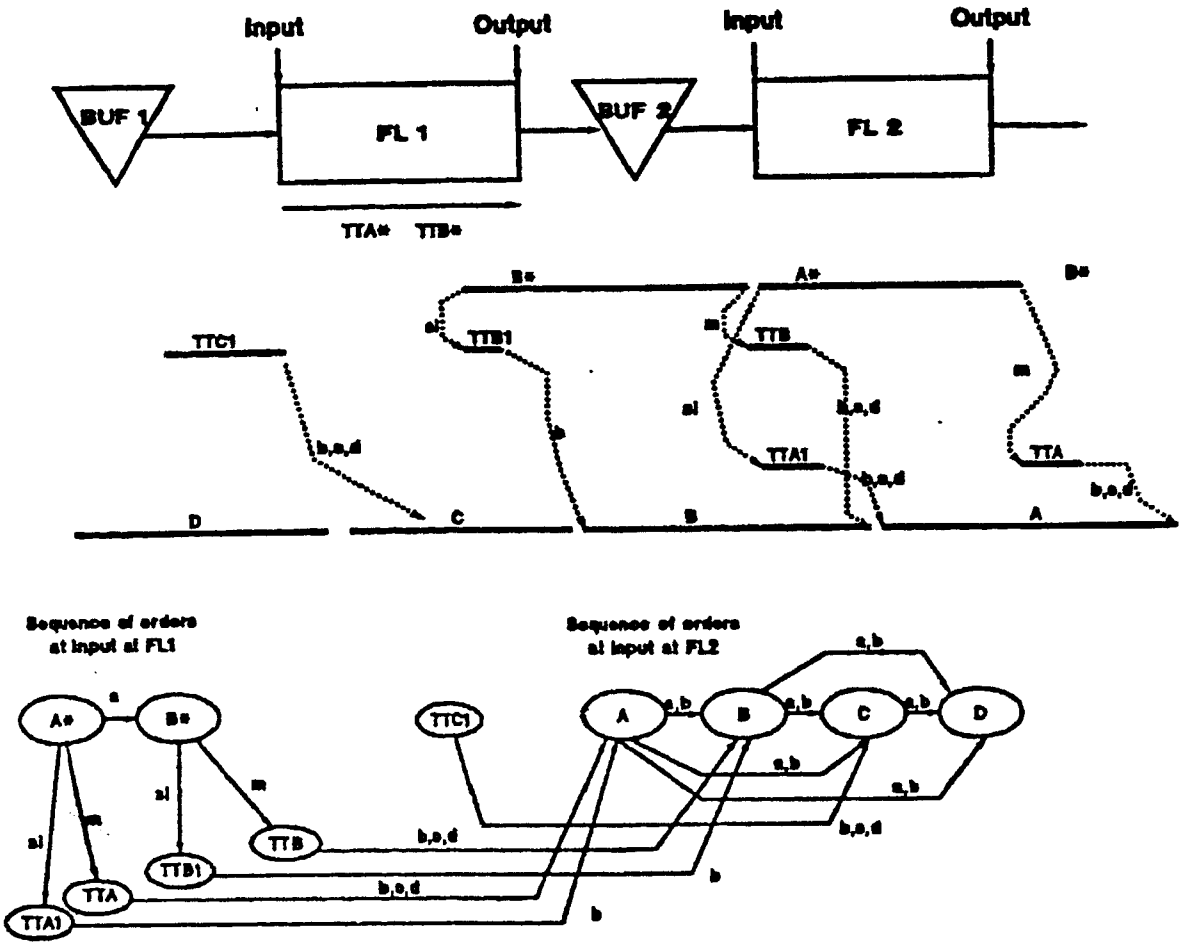


Figure 13: Test Case 4 (Synchronization of Two Coupled Flow Lines)

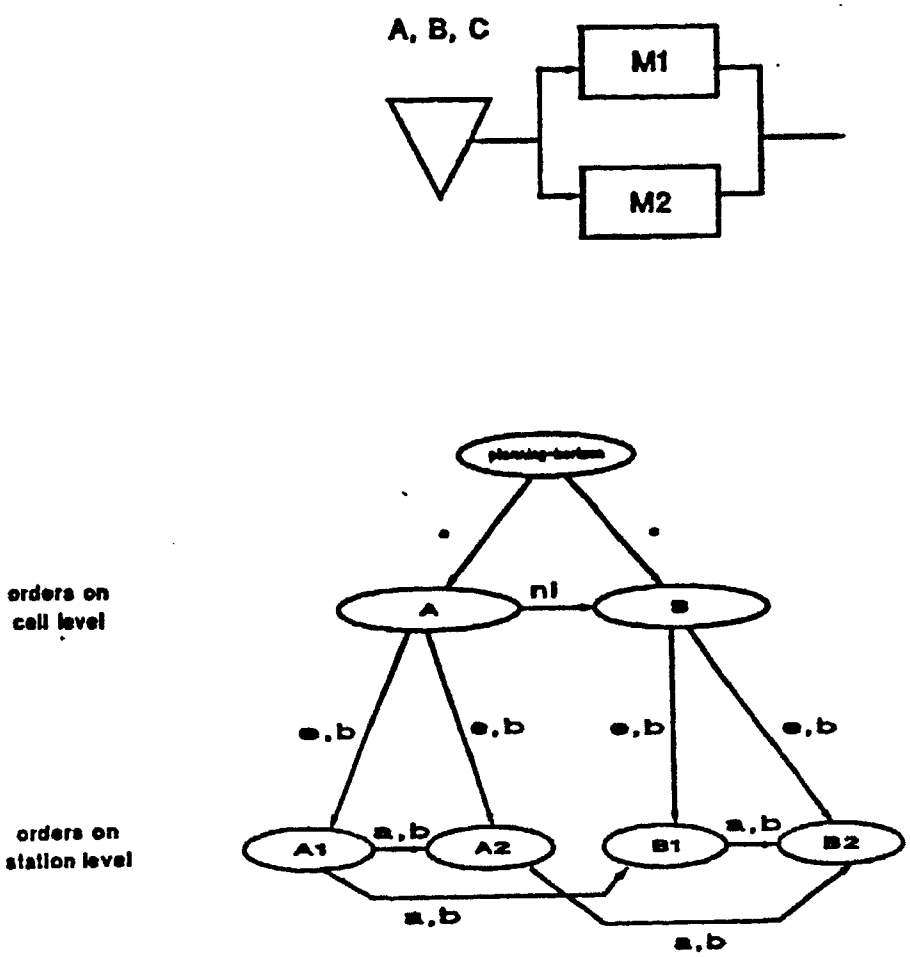


Figure 14: Test Case 5 (Two Machine Problem)