Large-Scale Temporal Data Bases for Planning in Complex Domains

Thomas Dean[1]
Department of Computer Science
Brown University
Providence, RI 02912

Abstract

Planning in any realistic setting requires the management of an enormous amount of information. This information is generally temporal in nature; prediction, plan choice, and debugging all involve reasoning about time. The assertions manipulated by traditional predicate-calculus data base systems, such as Prolog, are timelessly true. In the temporal data base system described in this paper, the classical data base assertion is replaced with the notion of a *tunc token.* For any given event or fact *type,* the data base will typically contain a large number of time tokens of that type. These tokens correspond to different occasions when an event of that type occurred or a fact of that type was made true and remained so for some period of time. This profusion of time tokens of the same type presents a problem for systems supporting temporal deductions of the sort needed in planning. Many routine planning operations must search through the data base for tokens satisfying certain temporal constraints. To expedite these operations, this paper describes a computational framework in which common-sense strategies for organizing temporal facts are exploited to speed search.

I. Introduction

Representing and reasoning about time play a critical role in many facets of everyday problem solving. We continually have to make reference to what has happened, is happening, and might possibly happen. To make matters more difficult, we have to cope with the fact that the world is constantly changing around us. To plan for the future we must be able to predict change, propose and commit to actions on the basis of these predictions, and notice when certain predictions are no longer warranted. All of this requires handling an enormous amount of complexly interdependent information.

In the past, the problems of efficient deduction and reason maintenance in predicate-calculus data bases used for planning have largely been ignored as researchers have grappled with the basic issues of reasoning about deadlines, iteration, conditionals, backtracking, and the like. As our representations have become more sophisticated and our ambitions to tackle more realistic domains have grown, the problems inherent in managing large temporal data bases have become a major factor limiting growth [12] [11]. What is needed is a computational framework in which common-sense strategies for organizing temporal facts can be easily exploited to expedite the search needed to support basic temporal inference procedures. As a simple example, suppose that you are planning a business trip and you are trying to remember if the travel agent has already confirmed your airline reservation. It shouldn't be necessary to recall (i e., search through) all of the events past, present, and future that involve your communicating with a travel agent. Only the most recent are likely to be of interest. Restricting attention to a particular time frame requires that facts that change over time are *indexed temporally,* that is to say, stored in such a way that facts and events common to a given time frame are easily accessible from one another. You would also like to avoid expending energy on facts that have nothing to do with travel agents or airline reservations. This requires that facts be *xndexed syntactically*.

Temporal indices, unlike their syntactic counterparts, are subject to frequent revision. Quite often the span of time associated with an event or fact is shifted, compressed, or expanded to suit a change of plans or reflect new information. Suppose that you decide to leave on your business trip a week earlier than previously planned. From this change, it should be apparent that certain prerequisite tasks (e.g., ordering plane tickets) must also occur earlier. Often, such a change will not require a major revision of plans. Where modifications are needed, however, the data base should clearly indicate just what parts of a plan have to re-thought and why (e.g., you might have to arrange for transportation to the airport in the event that the earlier departure will upset plans to share a ride with a friend).

To make accurate predictions about the future it is important to have access to information about the past. Information about the past helps us to understand what went wrong in situations where our actions fail to achieve their intended effect or our predictions fail to coincide with observation. Information about the past also helps us to resume interrupted or suspended tasks. Even if we were to reconcile ourselves to throwing away all record of the more distant past, we still have to consider that there are

[1] This work was supported in part by the National Science Foundation under grant IRI-8612644 and by an IBM faculty development award.

thousands of facts that we know are currently true and that are likely to figure in guiding our behavior. In addition, we generally keep track of a great many facts corresponding to long and short term goals and to predictions about the near and distant future. In any reasonably complex domain, the number of temporally dependent facts necessary for minimal competence is large. This paper describes how to go about organizing and maintaining these facts.

II. Thesis

[4] describes an approach to reasoning about time and a temporal data base management system that supports temporal reasoning of the sort necessary for planning. The temporal data base is called a *time map* [8], and the data base system is called a *time map manager* or TMM. In the TMM, the classical data base assertion is replaced by the notion of *time token* corresponding to a particular interval of time during which a general *type* of occurrence (a fact or event) is said to be true. For any given fact or event type, the time map will typically include many tokens of that type. The most expensive operation distinguishing temporal data base manipulations from those performed by static data base systems (e.g., Prolog) involves finding tokens that satisfy certain temporal constraints[2]. This operation, which we will call *token retrieval,* is the temporal analog of fetching assertions in the data base that match a given pattern. Token retrieval requires the system to search through the data base for time tokens whose type matches a given pattern and whose associated interval spans a specified reference interval. To support backward chaining efficiently, token retrieval should be demand driven: a call to the token retrieval routine should return one token (if possible) and a suspended procedure call (or *continuation)* which, if resumed, will supply additional tokens by taking advantage of the effort expended in finding previous tokens. To maintain integrity, all operations on the data base, including token retrieval, should supply succinct descriptions of the reasons why they are to be believed. This allows that all deductions performed by the system can be made contingent upon continued belief in the reasons supporting their component steps.

The primary claim of this paper is that, in many common situations, temporal data bases can be organized so that the cost of token retrieval is comparable with assertion retrieval in static data bases. Restricting our attention to the problem of retrieving a single token of ground type[3] *P* spanning the interval /:

1. In situations in which the set of tokens can be partitioned into temporally distinct periods (e.g., months, years, factory work shifts), the cost of token retrieval is proportional to the sum of:
   (a) the number of periods separating the period *Porigin* containing the beginning of / and the first period *Pdestination* preceding the beginning of *I* containing a token of type P, and
   (b) the cost of determining if any token of type *P* in *Pdestination* Spans *I.*
2. In situations in which the data base can be organized according to hierarchies of constraints (e.g., constraints between a task and its sub tasks), and these constraints are metric and reasonably tight[4], then the cost of determining if any token of type *P* in a given period spans / is proportional to the number of tokens of type *P* within that period.

As with any scheme for speeding up retrieval, there is a cost associated with organizing the data to support these fast retrieval routines. Fortunately, much of the work required for organizing time tokens is already being handled by other routines in the time map. Whenever a new constraint is added or removed, the system has to determine how the changes affect the current set of beliefs. This is the basic function of *temporal reason maintenance,* and it is essential for almost any interesting application of temporal reasoning [4]. The process of detecting changes in the set of beliefs is performed by propagating constraints through the constraint network. Essentially the system tries to compute new estimates (tighter bounds) on the distance in time separating selected pairs of points in the time map. These new estimates, referred to as *derived constraints,* can be used to either license new deductions or undermine old ones. The (derivation) complexity of a derived constraint is proportional to the length of the shortest path through the constraint network from which the associated distance estimate can be computed. If we assume that there is some bound, call it *maxderivedlength,* on the complexity of derived constraints necessary to catch all crucial changes in the set of beliefs, then we can perform constraint propagation in $0(m^3_{pt} \log m_{pt})$, where *pt* is one of the points being constrained and m_{pt} is the number of points reachable from *pt* by paths of length < *maxderivedlength.* In practice, m_{pt} is generally quite reasonable.

The organizational schemes described in this paper are integral with the basic functionality of the TMM; they instigate reorganization correctly in response to the addition of new information or the deletion of old, and they serve to expedite the basic operations used in temporal reason maintenance. The techniques involve methods for partitioning the set of time tokens both temporally and syntactically and for caching selected derived constraints and noticing when certain distance estimates become licensed by the current set of constraints or cease to be so. If we accept that the basic operation of temporal reason maintenance is essential, then the additional overhead in time and space necessary to implement these organiza-

[3] A *temporal* constraint is one that in some way restricts the order in which two points occur or the distance in time separating two points.

[3] A type *P* is *ground just* in case *P* contains no variables.

[4] In the framework being considered here, the distance between two points is represented by an upper and lower bound. By *tight* we simply mean that the difference between these bounds is small.

tional techniques is just a small constant factor times the number of tokens stored in the data base.

Complexity measures are generally misleading indicators of expected performance. The basic problem of representing temporal information using linear constraints has been studied by a number of researchers [7] [4] [11] [3]; most of the proposed techniques involve some sort of fairly straightforward polynomial-time algorithm. Unfortunately, even n^3 is out of the question for large n, where n corresponds to the total number of points or intervals being considered, and our experience [10] indicates that n will typically be on the order of several thousand. As soon as you add disjunction, the problem becomes *NP-complete* [13], but, again, asymptotic complexity doesn't tell the whole story. In the absence of a thorough analysis of the sorts of inference supported by a temporal reasoning system, evaluating such a system is impossible. In [4], we provide a detailed account of the inferential capabilities of the TMM, including special purpose techniques for handling disjunctions, default rules, and both antecedent and consequent reasoning. In this paper, we are primarily interested in the techniques required to expedite retrieval in large temporal data bases. We take as given that token retrieval cannot be a function of the size of the entire data base.

III. Temporal Data Base Management

The TMM consists of:

1. A data base (called a *time map)* that captures what is known about events and their effects over time. In particular, time maps are used to record information about the truth of propositions that change over time.
2. A query language that enables application programs to construct and explore hypothetical situations. This language supports simple queries of the form, "Is *P* true at time r?", as well as more complicated queries of the form, "Find an interval satisfying some initial constraints such that the conjunction (and P^1 ... P_n) is true throughout that interval."
3. A set of techniques for extending the information in the data base. These techniques allow for predictions on the basis of temporal antecedent conditions. Predictions added to the data base in this way are made to depend upon the antecedent conditions in a meaningful way.
4. A mechanism for monitoring the continued validity of conditional predictions. This mechanism extends the functionality of reason maintenance systems [6] to temporal domains.

A time map is a graph. Its vertices refer to *points* (or *instants)* of time corresponding to the beginning and ending of events. One point is related to another using *constraints* where a constraint is represented as a directed edge linking two points. Each edge is labeled with an upper and lower bound on the distance separating the two points in time. These bounds allow us to represent incomplete information concerning the duration and time of occurrence of events (e.g., unloading the truck will take between 20 and 25 minutes). Any two points can be related by finding a path from one point to the other, where a path from pt_0 to pt_n is just a sequence $ptoclpt_1$ ... c_npi_n such that *pto* through pt_n are points and c, is a constraint relating *pt-* to *pti.* New constraints are added to the time map by making assertions of the form (elt (distance *pt\ pt2) low high)* where this is meant to indicate that the quantity corresponding to (distance *pt\ pi2)* is an element of (•lt) the closed interval *low* to *high.*

An *interval* is just a pair of points. A *type* is denoted by a formula like (location obj73 locl4) or (move obj73 locl4 locl7). A *time token* (or simply *token* in situations where it should cause no confusion) is an interval together with a type, (begin *tok)* and (end *tok)* denote the begin and end points the time token *tok.* Predications of the form (occurs *type token-name)* are used to create new time tokens and refer to existing time tokens of a given *type.* The *token-name* gives us a handle so we can speak about the interval associated with a particular time token.

IV. Indexing Time Tokens

In this section, we will be concerned primarily with strategies for discriminating on data in order to expedite retrieval. A discrimination corresponds to a question (or deduction) concerning the form or content of the data. On the basis of the answer to such a question, the data is usually partitioned into disjoint sets so that if a program attempts to retrieve an item whose content depends on the answer to this question, then the system will know exactly where to look. This process of discriminating on data, asking questions and then partitioning according to the answer, can be thought of as caching the results of deductions that are likely to be frequently needed. To be useful, discriminations should substantially reduce search with a minimum overhead. Not all discriminations can be depended upon to remain valid as the data changes over time. Where the data is subject to change, there is an additional expense involved in keeping track of valid deductions.

The information content of a time token corresponds to the syntactic form of the token's type and the temporal extent (or scope) of the associated interval of time. Deductions corresponding *to* syntactic discriminations on the type of time tokens are never invalidated (though their utility may be undermined as tokens are removed from the database). All time tokens are indexed through what is called a *discrimination tree* or *dtree* [2]. Each nonterminal node in a dtree corresponds to a discrimination: a question whose answer determines which subtree various data items are stored in. Each terminal node in a dtree corresponds to a set (or *bucket)* of data items determined by the discriminations on the path leading from the root of the dtree to the terminal node.

Discrimination is demand based in the TMM. If the size of a bucket exceeds some fixed threshold, the TMM will attempt to subdivide the bucket by adding an additional discrimination node and some number of terminal nodes as dictated by the chosen partitioning scheme. If it is possible, partitioning a bucket of tokens is based upon a syntactic discrimination according to the types of the tokens stored in the bucket. If further syntactic discrimination is either impossible or undesirable, the system attempts to discriminate on the basis of the temporal scope of tokens. Temporal discrimination in the TMM involves choosing a temporal partitioning scheme and subdividing the overly large bucket of tokens according to this scheme. In the TMM, the application program is required to supply a set of hierarchically arranged temporal partitions of a time line (i.e., the real numbers) such that all of the partitions can be related via a single global frame of reference (i.e., 0). Attempts to *derive* an adequate partitioning scheme solely on the basis of the current contents of a bucket have proven difficult [9]. The partitioning scheme chosen must essentially anticipate the sort of questions that will frequently be asked during token retrieval. In the factory domain, the partitions supplied by the FOHBIN planner [10] correspond to weeks, days, eight-hour work shifts, and one-hour intervals. The system discriminates as demand dictates, starting with the coarsest partitions and refining only as required.

The partitioning scheme described above corresponds to a set of successively finer partitions of time with respect to a single clock. This is enormously useful as a coarse grained filter. Unfortunately, many events cannot be tied to a precise time, though they can be related precisely to one another. For instance, in reasoning about a chemical process, you may not know exactly when a catalyst was added to a reactor vessel, but you do know that within 8 to 10 minutes following the addition of the catalyst the reaction was complete. The TMM provides a mechanism for specifying hierarchies of event relations that can serve to guide search in determining temporal orderings among points that are not precisely known with respect to the global frame of reference of the partitioning scheme. The most common strategy involves the use of the event/subevent hierarchy. If e_1 is specified as a subevent of e_2, then the TMM can guarantee (within certain limitations) that there exists an edge in the time map connecting the beginning of e_1 and the beginning of e_2 labeled with the best bounds on the distance in time separating the two points.

These edges constitute cached deductions and are handled by the same mechanisms used in [4] to ensure correct behavior with regard to the addition and removal of information. The token retrieval machinery takes advantage of these cached deductions to speed search in determining the relative ordering of tokens which are not distinguished in the dtree by temporal discriminations. These combined searching and caching techniques guarantee that under certain conditions[5] the machinery for determining the bounds on the distance between two points will always return the best bounds and will do so in time proportional to the depth of the hierarchy. It is also possible to show that the system never reports false bounds and that the behavior of the system degrades gracefully as the information becomes less precise and the *maxderivedlength* increases beyond the fixed threshold. For most problems the depth of the hierarchy is seldom greater than 20 and the alternative exhaustive search would cost on the order of n^3 where *n* is the total number of tokens in the partition (often on the order of several hundred).

Token retrieval routines use the dtree to provide a set of candidate tokens and then determine the relative ordering of the beginning of these tokens using the search methods described in the previous paragraph. Determining the duration of a fact token relative to a reference interval is also accomplished using search methods that exploit the hierarchical partitions and cached distance estimates. All the search routines return the information requisite for setting up appropriate data dependencies. Searches corresponding to different token retrieval requests can be coroutined to support efficient backtracking during backward chaining.

V. Algorithmic Details

The TMM employs a heuristic graph traversal routine to compute bounds on the distance separating pairs of points in the time map. These bounds are used to determine relations between pairs of points and intervals. Estimates of the distance between pairs of points are computed by finding paths through the network of constraints. Recall that a path in the time map is a sequence of points and directed edges corresponding to constraints. Each edge *c* is labeled with an upper and lower bound, denoted Low(c) and HIGH(c) respectively. For each path $p = pt_0 c_1 pt_1 \ldots c_n pt_n$ we have BOUNDS(p) = *(low, high)* where *low* = $\sum_{i=1}^{n} \mathrm{LOW}(c_i)$, and $high = \sum_{i=1}^{n} \mathrm{HIGH}(C_i)$. In computing the best bounds, the heuristic graph traverser tries to find the paths with the greatest lower and least upper bounds. The details of the TMM's graph traverser are described in [4], and won't be repeated here. For a discussion and analysis of existing constraint propagation techniques for applications in artificial intelligence see [3].

In addition to asserting constraints between pairs of points corresponding to the begin and end of time tokens, it is also possible to constrain the begin or end of a time token with respect to the global frame of reference (mentioned in the previous section). Figure 1 shows a time map and the time line corresponding to a fixed global frame of reference; five tokens (notated $T_1 \ldots T_5$) and their corresponding beginning points *(pt\ ... pt3)* are labeled for easy reference. Constraints are depicted as curved lines (e.g., those labeled C_1 and C_2). The points in the shaded area

[5] The set of tokens must be totally ordered and the *maxderivedlength* (see Section II.) must be fixed.

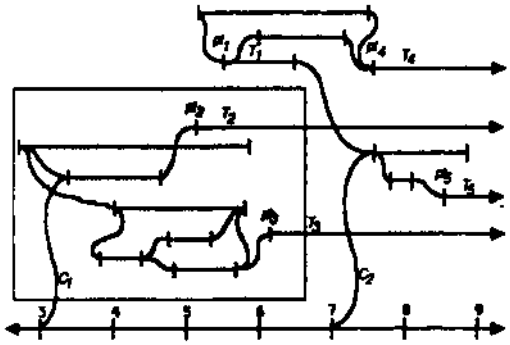


Figure 1: Time map with privileged frame of reference

(e.g., p_{t2}) can be related to points outside the shaded area (e.g., pt_1 or pr_4) only through the constraints C_1 and C_2 and the point in time corresponding to the frame of reference of the global time line. Given that one or both of $T2$ and T^3 are of type P, paths through the global frame of reference will be important in determining if P is true throughout the interval associated with $T1$. For relating pairs of points isolated in the same cluster of tokens (e.g., pt_1 and pt^3), the global frame of reference may not help in establishing the necessary distance estimate, but, as we will see, the global frame of reference provides an invaluable service by allowing the search routines to ignore large portions of the time map.

In order to expedite token retrieval, the TMM precomputes and stores (i.e., *caches)* certain point-to-point distance estimates. Caching is performed as part of the constraint propagation routines used to implement tern poral reason maintenance. The TMM can maintain an estimate, guaranteed exact under certain assumptions (see [4]), of the distance in time separating selected pairs of points. Obviously, it would be wasteful to cache distance estimates for all pairs of points. Selective caching, on the other hand, can provide real benefits. In Section VI., we will see how caching estimates of the distance between each point and the global frame of reference forms the basis for an effective temporal discrimination scheme. In [5], we demonstrate how a strategy for caching distance estimates between points corresponding to related tokens can expedite search within portions of the time map that are not highly constrained with respect to the global frame of reference.

VI. Hierarchical Partitioning Schemes

In the following, let R denote the set of real numbers, and $[a, b)$ the interval defined by $\{x \mid x \in R, a \leq x < b\}$. For our purposes, a *partition* $\mathcal{P}$ of an interval I (subset of R) is just a set $\{[x, y) \mid x, y \in I\}$ such that $I_1 \cap I_2 = \emptyset$ for all distinct I_1 and I_2 in $\mathcal{P}$, and for each $x \in I$ there exists an I' in $\mathcal{P}$ such that $x \in I'$. A *hierarchical partitioning scheme* consists of a sequence of partitions $\mathcal{P}_0\mathcal{P}_1 \ldots \mathcal{P}_n$ of R such that for each $i < n$ if $I \in \mathcal{P}_i$ then there is a set intervals in $\mathcal{P}_{i+1}$ that partitions I. In addition, we insist that $\mathcal{P}_0$ is always the set consisting of just R. $\mathcal{P}_i$ is said to

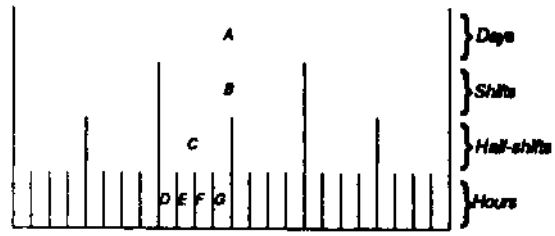


Figure 2: Portion of a strict hierarchical time-line partition

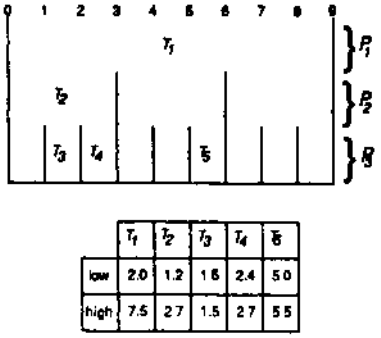
be more *restrictive* than $\mathcal{P}_j$ with respect to / just in case $I \in \mathcal{P}_j$ and there is a subset S of p, such that $|\mathcal{S}| > 1$ and 5 partitions J. A hierarchical partitioning scheme is *strict* if it is the case that for any i such that $0 \leq i < n$, if $I \in \mathcal{P}_i$, then $\mathcal{P}_{i+1}$ is more restrictive than p, with respect to /. All the partitioning schemes we will be looking at in this paper are strict.

A *time-line partition* is just a partition of R such that 0 is identified with a particular frame of reference (e.g., midnight on April 30, 1777) and each real number x corresponds to an offset in time from this global frame of reference as measured by a particular clock. The global frame of reference simplifies internal bookkeeping and provides a basis for using dates in specifying constraints. Figure 2 shows part of a strict hierarchical time-line partition in which the partitions consist of weeks, shifts (eight hour periods), half-shifts (four hour periods), and one hour periods offset from a fixed zero point. In Figure 2, C is contained in A and B, and is partitioned by {D,E,F,G}.

Specifying constraints with respect to the global frame of reference is made particularly easy using *dates.* A *date* is just an offset from the global frame of reference specified in terms of the current partitioning scheme. For instance, in the scheme mentioned in the previous paragraph, the date ((veeks 2) (days 3) (shifts 2) (hours 1) (minutes 15))[6] is easily converted into an offset in minutes from the current global frame of reference. Often, it is convenient to specify a default date (e.g., noon today) and then specify offsets, called *reldates,* relative to the default. Dates and reldates can appear anywhere in a formula that a point can. As an example, if the default date is noon today, asserting (eit (distinct (begin task41) (reldate (hours 2) (minutes 30))) -10 10) determines that task41 begins between 2:20 and 2:40 this afternoon. The TMM employs simple rewrite rules to translate between various internally used partitioning schemes and computationally cumbersome (but familiar) methods of dating based upon the modern calendar.

Every point is identified with a tuple *(low,high),* called its relative offset, indicating the best (lower and

[0] Partition divisions not mentioned in a date default to 0.



	T_1	T_2	T_3	T_4	T_5
low	2.0	1.2	1.6	2.4	5.0
high	7.5	2.7	1.5	2.7	5.5

Figure 3: Data for demonstrating temporal indexing

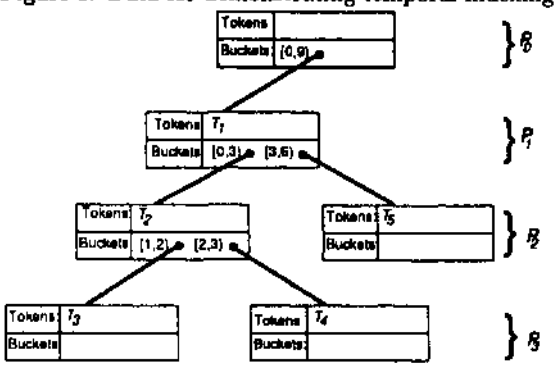


Figure 4: Portion of a temporal discrimination tree

upper) bounds over all paths through the network of constraints on the distance in time separating the point from the global frame of reference. Relative offsets are updated during constraint propagation. The constraint propagation routines ensure that, if additions or deletions to the set of constraints require that the relative offset of a point be updated (i.e., the bounds made either more or less restrictive), then the system can easily detect this and respond appropriately. The relative offset determines how tokens are indexed in the TMM discrimination tree. To simplify the discussion of temporal indexing, we will assume that all tokens are syntactically indexed down to atoms according to type, and consider only those nodes in the discrimination tree corresponding to temporal indices. A temporal index is implemented as a data structure called a TBUCKET consisting of a set (possibly empty) of tokens that can't be further discriminated upon, a partition interval [a,b), and a set of subindices (i.e., TBUCKETs) sorted by their associated partition intervals. For efficiency reasons, subindexing is usually postponed until the set of tokens stored at an index exceeds some fixed threshold. Figure 3, shows a simple hierarchical partitioning scheme and a table indicating the relative offsets for five tokens of the same type. Figure 4 shows a portion of a discrimination tree. (Recall that $\mathcal{P}_0$ is the singleton set corresponding to the entire time line.) Note that T_5 could be further discriminated, but is not in this case since the TBUCKET containing it would otherwise be empty. If the relative offset of the beginning of T_2 was changed from $\langle 1.2, 2.7 \rangle$ to (1.2,1.8), then T_2 would be further discriminated ending

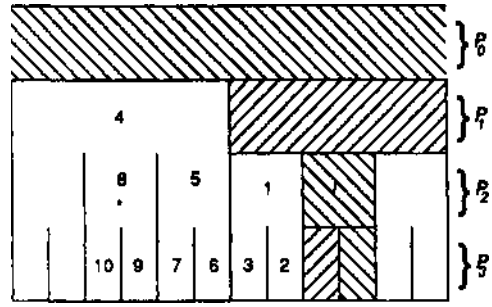


Figure 5: Search in hierarchical partitions

up in the same bucket with T_3. If, on the other hand, the relative offset was changed to (1.2,3.5) or (1.2, +oo), then T2 would end up (respectively) in the same bucket with T_1 or in the top-most index corresponding to p_o .

Now, it is straightforward to describe the algorithm used in the TMM for token fetching during temporal backward chaining. The algorithm makes use of two sorts of indexing in performing the requisite search: direct-path indexing and indexing relative to the global frame of reference. The steps in the algorithm are:

1. Determine a partition *V* and an interval *I* belonging to *V* such that the fetch interval is constrained to (necessarily) begin during I and cannot be shown to (necessarily) begin during any interval belonging to a partition more restrictive than *V*.

2. Using the heuristic graph searching routines, try each matching token whose associated partition interval either is contained in I or contains J. In Figure 5, the shaded areas correspond to the partition intervals searched during this step. If a token is found that either begins before or can be constrained to begin before the fetch interval, then an attempt is made to determine whether or not that token persists throughout the fetch interval. Relative offsets and the heuristic search routines are used to determine the relative ordering of the end point of the found token and the end point of the fetch interval.

3. If the previous step fails to find an appropriate token or set of tokens then we search through the remaining tokens of the desired type as follows:

   (a) Set the variable *early-termination* to false.

   (b) Determine a partition interval, call it N, preceding *I* such that there is no other unexamined partition interval preceding / which is either later than N or in a less restrictive partition.

   (c) For each matching token in AT, use the relative offset from the global frame of reference to determine if the token persists long enough.

   (d) Mark N as examined.

   (e) If a matching token is found that fails to persist throughout the fetch interval because it has been

clipped by a contradictory token, then set *early-termination* to true,

(f) If *early-termination* is true and N is an element of the most restrictive partition, then stop, otherwise return to step 3b.

Figure 5 illustrates the order in which partition intervals are examined during step 3 of the fetch. The starred partition interval is meant to indicate where the variable *early-termination* was set to true.

The above indexing scheme relies heavily upon their being a single global frame of reference and a rather simple and restrictive hierarchy of partitions. The fact that the partitions are strictly nested (i.e., for each $i < n$, if $I \in \mathcal{P}_i$, then there is a set of intervals in P,+j that partitions *I)* can result in certain inefficiencies at the boundaries of partition intervals. Points that are slightly unordered with respect to a major time break will be assigned a fairly unrestrictive partition interval even though their relative offset is known with considerable precision. For instance, an event constrained to begin at midnight January 1 give or take a minute will end up in the bucket corresponding to the decade partition, assuming a partition according to decades, years, months, weeks, days, and hours. Such events cause the system to do a bit more work, but since they are relatively rare, the overall effect is negligible. Tokens whose begin points are known with some precision (i.e., the difference between the lower and upper bound of the relative offset is small), but which, nevertheless, are unordered with respect to major time breaks might be handled by considering pairs of partition intervals adjacent to the time break and indexing the token in more than one interval, but no attempt has been made to implement such a strategy in the current system. Similarly, the beginning of fetch intervals can also span major time breaks and for this reason it is often useful to split the fetch and perform the requisite search on a case-by-case basis.

VII. Related Work

The techniques described in this paper have profited from many sources. [9] describes a discrimination network capable of indexing spatial objects on the basis of metric information. Methods for organizing intervals hierarchically have appeared quite frequently in the literature [1] [7]. There have also been a number of strategies suggested for guiding search in reasoning about time [12] [11]. A careful reading of the discussion of constraint propagation techniques in [3] convinced us that, while the general problem of reasoning in large constraint networks is hopeless, in most practical situations involving time, the requisite search can be directed with amazing precision.

VIII. Conclusion

The TMM provides a wide range of functionality (backward and forward temporal inference, dependency directed default reasoning, temporal reason maintenance) in a simple-to-use system (predicate-calculus syntax and PROLOG compatibility) in which routine temporal reasoning is optimized using sophisticated caching and search techniques to speed inference. Straightforward partitioning schemes supplied by an application program are used to fragment a temporal data base into non-overlapping periods. Under situations that arise frequently in everyday reasoning, token retrieval, the basic operation common to most forms of temporal inference, can be performed in time proportional to the sum of (a) the number of periods separating the period containing the beginning of the reference interval and the first previous period that contains a token of the desired type, and (b) the number of tokens of that type beginning in that previous period. The result is that the performance of the temporal inference engine corresponds roughly to our expectations given the distribution of tokens of the underlying fact types being manipulated. The discussion of techniques in this paper is necessarily cursory. A technical report [5] provides additional detail concerning both the algorithms and their expected performance.

References

1. Allen, James, Maintaining knowledge about temporal intervals, *Communications of the ACM* 26 (1983).
2. Charniak, Eugene, Flies beck, Christopher K. and McDermott, Drew V., *Artificial Intelligence Programming* (Lawrence Erlbaum Associates, 1980).
3. Davis, Ernest, *Constraint Propagation of Real-Valued Quantities* $_y$ Technical Report 189, New York University Department of Computer Science, 1985.
4. Dean, Thomas, and McDermott, Drew V., Temporal Data Base Management, *Artificial Intelligence* 32 (1987).
5. Dean, Thomas, *Large-Scale Temporal Data Bases,* Technical Report CS-86-15, Brown University Department of Computer Science, 1986.
6. Doyle, Jon, A truth maintenance system, *Artificial Intelligence* 12 (1979).
7. Malik, Jitendra, and Binford, Thomas 0., Reasoning in Time and Space, *Proceedings IJCAI 8, Karlsruhe, West Germany,* IJCAI, 1983.
8. McDermott, Drew V., A temporal logic for reasoning about processes and plans, *Cognitive Science* 6 (1982).
9. McDermott, Drew V. and Davis, Ernest, Planning routes through uncertain territory, *Artificial Intelligence* 22 (1984).
10. Miller, David P., Firby, R. James, Dean, Thomas L., Deadlines, Travel Time, and Robot Problem Solving, *Proceedings IJCAI 9, Los Angeles, Ca.,* IJCAI, 1985.
11. Smith, Stephen F., *Exploiting Temporal Knowledge to Organize Constraints,* Technical Report CMU-RI-TR-83-12, Carnegie-Mellon University Intelligent Systems Laboratory, 1983.
12. Vere, Steven, Temporal Scope of Assertions and Window Cutoff, *Proceedings IJCAI 9, Los Angeles, Ca.,* IJCAI, 1985
13. Vilain, Marc, Constraint Propagation Algorithms for Temporal Reasoning, *Proceedings AAAI-86, Philadelphia, Pa.,* AAAI, 1986.