INTEGRATION OF OPTIMAL SCHEDULING WITH CASE-BASED PLANNING

General Electric Company

Sponsored by
Advanced Research Projects Agency
ARPA Order No. 7686

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York
This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-95-147 has been reviewed and is approved for publication.

APPROVED:

DONALD F. ROBERTS
Project Engineer

FOR THE COMMANDER:

JOHN A. GRANIERO
Chief Scientist
Command, Control & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (C3CA) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.
INTEGRATION OF OPTIMAL SCHEDULING WITH
CASE-BASED PLANNING

P. Bonissone
J. Stillman
J. Aragones
R. Arthur
S. Ayub
J. Farley
L. Blau Halverson

Contractor: General Electric Company
Contract Number: F30602-91-C-0030
Effective Date of Contract: 15 May 1991
Contract Expiration Date: 15 July 1994
Short Title of Work: Integration of Optimal
Scheduling with Case-Based Planning
Period of Work Covered: May 91 - Jul 94
Principal Investigator: Pierro Bonissone
   Phone: (518) 387-5155
RL Project Engineer: Donald F. Roberts
   Phone: (315) 330-3577

Approved for public release; distribution unlimited.

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by Donald F. Roberts, RL (C3CA), 525 Brooks Rd, Griffiss AFB NY 13441-4505.
This report combines information from several articles and theses. Those of particular interest are summarized here. "Planning in an Uncertain and Dynamic Environment with Weak Domain Theory" summarizes the author's research and results in the field of planning in the Mergers and Acquisitions (M&A) domain, as well as giving a comprehensive history of CBR and CBP. "Similarity Measures for Case-Based Planning Systems" focuses on the case retrieval problem and the computation of similarity measures between cases. "Planning with Dynamic Cases" describes the Case Representation Language (CRL) and the architecture of a Case-Based Planning (CBP) system. "Representing Cases and Rules in Plausible Reasoning Systems" describes a hybrid system that integrates Case-Based Reasoning (CBR) and Rule-Based Reasoning (RBR) systems. "Tachyon: A Constraint-Based Temporal Reasoning Model and Its Implementation" provides an overview of the Tachyon temporal's reasoning system and discusses its possible applications. "Dual-Use Applications of Tachyon: From Force Structure Modeling to Manufacturing Scheduling" discusses the application of Tachyon to real-world problems, specifically military force deployment and manufacturing scheduling. A Case Study in Integration of Case-Based and Temporal Reasoning using CAFE and Tachyon" describes the integration of CAFE (a Case-Based Tool for Expansion of Forces) with Tachyon, with the goal of allowing the user to tailor (see reverse).
forces for a current mission by using historical cases, while also tracking the effect of temporal constraints on those forces.
## Contents

### I Case Based Reasoning and Case Based Planning

1 Background and History
   1.1 Case Based Reasoning ................................................. 5
      1.1.1 Previous Work .................................................. 6
   1.2 Case Based Planning .................................................. 7

2 Case Memory
   2.1 Organization of Case Memory ......................................... 9
      2.1.1 Conceptual Knowledge ........................................... 9
      2.1.2 Episodic Knowledge ............................................ 10
   2.2 Case Representation Language ....................................... 12
      2.2.1 Domain Knowledge Representation .............................. 13

3 Case Retrieval
   3.1 Analysis of Cases .................................................... 19
   3.2 Feature Value Comparisons ......................................... 22
   3.3 Combination of Similarities ........................................ 25

4 Plan Development
   4.1 Plan Extraction ....................................................... 32
      4.1.1 Representation of plan ......................................... 32
      4.1.2 Identification of a plan ....................................... 32
      4.1.3 Identification of resources and constraints .................. 35
   4.2 Planning Architecture .............................................. 37
   4.3 Goal Resolution ..................................................... 39
      4.3.1 Identifying an event for a goal ............................... 40
      4.3.2 Identifying an interpretation for a goal ..................... 41

5 Evaluation and Results
   5.1 Methodology for Evaluating the Generated Plans ................. 42
      5.1.1 Generate plans for an agent .................................. 42
      5.1.2 Represent the generated plan as a case ..................... 42
      5.1.3 Compare the events in the planned case and stored case .... 43
   5.2 Interpretation ...................................................... 45

6 Conclusions and Summary .................................................. 46
## II Temporal Reasoning

<table>
<thead>
<tr>
<th>7 Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1 Motivation and Applications</td>
</tr>
<tr>
<td>7.2 Background</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8 Design Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1 Defining the Problem</td>
</tr>
<tr>
<td>8.2 Difficulties</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>9 Results/Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1 Temporal Constraint Networks</td>
</tr>
<tr>
<td>9.2 The Tachyon Model</td>
</tr>
<tr>
<td>9.3 Propagation of Constraints</td>
</tr>
<tr>
<td>9.4 Path Consistency</td>
</tr>
<tr>
<td>9.5 Interval Trees</td>
</tr>
<tr>
<td>9.6 The Tachyon User Interface</td>
</tr>
<tr>
<td>9.6.1 Objectives</td>
</tr>
<tr>
<td>9.6.2 Functionality</td>
</tr>
</tbody>
</table>

| 10 Discussion & Related Issues | 77 |
| 11 Future Directions and Conclusions | 78 |
| 12 Project Planning Example | 80 |
| 13 Scheduling Example | 82 |

## III Integration of Case Based Reasoning and Temporal Reasoning

<table>
<thead>
<tr>
<th>14 Background</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.1 CAFS/CAFE</td>
</tr>
<tr>
<td>14.2 Tachyon</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>15 Integrated Capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.1 An Example</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>16 Future Directions &amp; Conclusions</th>
</tr>
</thead>
<tbody>
<tr>
<td>16.1 ForMAT Integration</td>
</tr>
<tr>
<td>16.2 Extended Capabilities of Tachyon</td>
</tr>
<tr>
<td>16.3 Conclusions</td>
</tr>
</tbody>
</table>
Part I

Case Based Reasoning and Case Based Planning
Chapter 1

Background and History

1.1 Case Based Reasoning

Case Based Reasoning (CBR) is at the core of any Case Based Planning (CBP) system. Obviously, the plans generated by a CBP system are highly dependant on the CBR foundation. As noted in [2]:

Case-Based reasoning (a method of analogical reasoning), thought common and extremely important in human cognition, has only recently emerged as a major reasoning methodology. Case-based reasoning (CBR) involves solving new problems by identifying and adapting solutions to similar problems stored in a library of past experiences/problems. The important steps in the inference cycle of CBR are to retrieve cases from the case library which are most relevant to the problem at hand and to adapt the retrieved cases to the current input. Within this broad framework, two major classes of CBR can be identified [97]: problem solving CBR and precedent based CBR. In problem solving CBR, the emphasis is on adapting the retrieved cases for finding a plan or a course of action to solve the input problem. Case-based planning is in the class of problem solving CBR. In precedent based CBR, the emphasis is on retrieving cases so as to justify an action or explain a solution. A common application of precedent based CBR is in legal domain [55, 58, 53].

A similar, more elaborate definition can be found in [57].

Case-based reasoning (CBR) is the process of using previously acquired solutions to problems as the basis for computing new solutions to new problems. The stored problem descriptions and solutions are cases. CBR has been applied to problem solving in many different application areas, for example legal [29, 47, 49], medical [85], financial [60] and engineering [46, 45].

Case-based reasoning can provide an alternative to rule-based expert systems, and is especially appropriate when the number of rules needed to capture an expert’s knowledge is unmanageable or when the domain theory is too weak or incomplete. Historically, CBR has shown its greatest success in areas where individual cases or precedents govern the decision-making processes, as in case law.

CBR Reasoning Process: In general, CBR systems comprise a case-memory, indexing, matching and retrieval mechanisms, and a reasoning component. The matching and retrieval mechanisms, driven by the current context (reasoner’s goal and probe), return the most
similar cases from the case memory. Similarity among cases is based on an evaluation of salient and relevant features. In some CBR systems the output of the matching process provides a complete solution to the input problem without requiring additional reasoning. In others, the reasoning component will process the retrieved cases, adapting their solutions (plans, explanations, interpretations) to apply in the current situation.

**Uncertainty in CBR:** Uncertainty and incompleteness pervade the CBR reasoning process. Uncertainty is present in the semantics of abstract features used to index the cases, in the evaluation of the similarity measures computed across these features, in the determination of relevancy and saliency of the similar cases, and in the solution adaptation phase.

Incompleteness is present in the partial domain theory used in the indexing and retrieval, in the (usually) sparse coverage of the problem space by the existing cases, and in the description of the probe.

### 1.1.1 Previous Work

We give a summary of the previous work done in this area in [59].

One of the earliest and best known examples of a case based planner is the CHEF system built by Kristian Hammond [72]. The CHEF program addresses the problem of planning in the cooking domain. It generates new plans (recipes) by adapting the sequence of actions from similar past plans (recipes). The input to CHEF is a list of goals (such as hot stir fry dish with chicken and broccoli), that have to be satisfied. The result of planning by CHEF is a plan that satisfies these goals. If part of a plan fails, CHEF repairs the plan and an index to the repair is added to memory to avoid repeating that planning failure. CHEF does not have any interpretation of input goals for the retrieval of plans but it does exhibit complex plan adaptation and learning capabilities.

CHEF retrieves similar cases based only on goal similarity. Therefore when a plan fails during execution, due to failed preconditions or objectional results, CHEF stores the failure, but must begin from scratch in rebuilding the plan (recipe). One major requirement of the problem domains which we are addressing is the ability to continue planning from any point in plan execution, while maintaining consistency with previous actions. This is the result of having to deal with other (possibly antagonistic) agents changing the world state.

More recent work in planning by Hammond et al.,[73] addresses the issues of opportunism and flexible plan use in the areas of reactive planning and strategic/tactical planning. In RUNNER, the observation of particular values of environmental features (state), triggers the activation of a goal(s), which is used to index into memory to retrieve an existing plan for satisfying the goal(s). The retrieved plan is used to give permission to sub-plans/actions to take place. An action must have both permission and opportunity to be executed. Opportunity for an action depends on the observation of particular features in the environment. In summary the guidance on permissible actions comes top down from the goals and recognition of opportunity comes bottom up from the state. The action which lies on their intersection is taken.

The representation of cases in Redmond’s work [94] is the closest to our approach to case representation. In his approach, cases are stored in pieces, or snippets [82]. Each snippet is organized around one goal and contains both local context (state/knowledge obtained from
the actions taken so far) and global context (the overall problem description). The pieces of a case (snippets) are linked to represent the whole case of problem solving. The underlying assumption in the architecture of snippets and this approach is that there is only one agent executing the plan/actions. The changes in the state of knowledge and the environment are due to the actions in the pursuit of a certain goal (around which snippet is organized). As it is now, the representation of snippets does not lend itself naturally to represent cases where the state of the world changes due to the actions of multiple agents.

Our previous work in the area of CBR includes MARS [60], a Mergers and Acquisition system chosen to illustrate the use of reasoning (Case Based and Rule Based Reasoning) for solving problems in a complex business domain, and CARS [30] a case based system for the same domain, which reasoned with a static representation of the events in a takeover, and explored the integration of independent case-based and rule-based systems. MARS was used to explore the possible contribution of previous cases to problem solving in a rule-based system. Cases were analyzed off-line and stored as plausible rule templates. CARS was used primarily as a precedent based system; when given a probe, and indication of the reasoners goals, it returned the most similar cases from the case library.

The domains which are currently the focus of our efforts are areas where multiple agents, with different goals and viewpoints, attempt to plan strategies using incomplete, or uncertain information. In addition, these cases develop over time, requiring us to reason about sequences of events. We found that the lack of a representation of the dynamic aspects of these cases severely limited our reasoning capability when we moved our work with CARS into the area of solution adaptation. Therefore we developed a case representation language which provides for the representation of the dynamic aspects of the cases.

1.2 Case Based Planning

CBP is a specialized application of CBR. First, CBR is used to retrieve and analyze similar cases. Then CBP algorithms are applied to generate a plan.

An explanation of the development of a Case Representation Language (CRL) with regards to Case Based Planning can be found in [2]:

Classical planning systems assume a good domain theory for generating plans. However, complex domains have incomplete domain theories. In some problem domains, lack of a good domain theory can be compensated by using past cases to guide the planning system. These past cases may contain uncertain information and may have evolved over time.

Case-Based Reasoning (CBR) uses past cases, which contain acquired solutions to previous problems, as the basis for computing new solutions to new problems. The CBR architecture consists of a case library and an inference cycle. The case library is an organized collection of previously experienced problems and their associated solutions. The inference cycle is an iterative procedure for solving the current problem. Its two major components are the retrieval of relevant cases and their adaptation to obtain a suitable solution.

For our CBR system, we have developed a Case-Representation Language (CRL) to store previous cases, a process to determine case similarity to identify the most appropriate case to retrieve, and a process to adapt a retrieved case to get a suitable plan for the goals.

The CRL is developed to represent cases that evolve over time and exhibit uncertain
information. It provides a way for representing cases in their natural evolution without many transformations or loss of information. It also allows the expert to add his own explanations to the case evolution.

The case similarity between the probe and stored cases is done by aggregating their situational and dynamic similarities. Situational similarity is obtained by determining the similarity between the states of the objects involved in the cases. Dynamic similarity is obtained by determining the similarity between the evolution of the cases. The aggregation is done hierarchically according to a semantic taxonomy.

The adaptation of a retrieved case is done by extracting a plan from the retrieved case. The plans for goals not resolved in the extracted plan are identified in cases in the case library. The extracted plan is augmented with the identified plans and is structurally adapted to the current situation. This plan is then modified, using the cases in the case library, to ensure its executability in the current context.

Our CBR system, named Combined Approximate Reasoning System (CARS), is tested in the domain of Mergers and Acquisitions (M&A). It uses combined reasoning to develop plans. Partial domain knowledge of M&A (i.e., financial knowledge) is represented using rules and its weak domain theory is complemented by real M&A cases.
Chapter 2

Case Memory

Case Memory is at the foundation of case retrieval. Without a good system for organization of case memory, useful retrieval of similar cases is nearly impossible.

2.1 Organization of Case Memory

Since the organization of case memories is central to any CBR system, there are numerous explanations of memory organization in the literature.

Our organization of case memory is described in [2]:

The case memory has been designed to represent cases consisting of the top-level goal(s) and information about states and events. This information can be obtained from two basic sources: world observers and domain experts. World observers are capable of recording the state at any time, and of recognizing the execution of state changing actions in the world. Domain experts are capable of interpreting/relating these states and actions to the behaviors of an agent(s) attempting to satisfy the top-level goal(s) of the case.

The case memory is organized around two types of knowledge:

- **Conceptual Knowledge** is the information about the objects, actions, and goals in the domain. This knowledge, which represents an incomplete domain theory, is used during retrieval, case comparison, and solution adaptation.

- **Episodic Knowledge** is the collection of cases. Each case is represented as a situation/solution pair where the situation consists of the top-level goal(s) and a starting state, and the solution consists of the representation of the observable portion of the agent’s execution of the plan to satisfy the goals.

2.1.1 Conceptual Knowledge

We summarize the definition of conceptual knowledge, and its application to the domain of M&A in [56].

The conceptual knowledge can be organized into various hierarchies depending on the problem domain. It provides a way to define various entities that are involved in a case. It also provides a channel for understanding entities in cases for various purposes.
In the M&A domain three hierarchies were used for representing the conceptual knowledge: object hierarchy, action hierarchy and goal hierarchy. These hierarchies are implicitly linked to each other and explicitly linked to the stored cases. Examples of links are: objects from one hierarchy are used as slot fillers or slot-type specifiers (implicit link), like instance of "common stocks" object is used as slot filler in an instance of "tender offer" action, and interpretations of some actions in a case are linked to a node in the goal hierarchy (explicit link)

2.1.2 Episodic Knowledge

The episodic knowledge of the system is a collection of instances of cases in the Case Base. We consider each case as the set of executed plans of one or more agents for achieving their top level goals from a given initial state. The parallel to a case in a classical generative planning paradigm is a state space representation of multiple plans of the agents for achieving some top level goals and a description of an initial state. [2]

The representation of dynamic cases using CRL in other domains like transportation is discussed in [59].

As discussed earlier, each case is represented by a network of events (actions taken) and a sequence of states in temporal order. Identifiable plan steps are represented using interpretations and these interpretations facilitate the indexing, understanding, and re-use of the plans. Links are used to encode the explanatory information about the relations between events and states. There are four types of links: causal, temporal, membership, and enable. Each link can be qualified by a degree of belief.

A partial representation of a case is given in Figure 2.1. In this case the action tender-offer by the raider company is followed by the actions reject-tender-offer and announce-restructure-plan by the target company. The initial state of the objects when the case begins is phase-1. The state of the world changes in state-2 with the increase in the price-per-share of the target company. The state of the world changes to state-3 when the target company knows for sure that it is the target of a hostile takeover. The sequence of state changes have a temporal order in which state-3 follows state-2. The change in the world from state-3 was certainly (i.e. belief in this causal relation is *certain*) caused by the action tender-offer. This action also caused an action reject-tender-offer by the target company. The new belief of the target company in state-3 enabled them to take a difficult action such as announce-restructure-plan. The actions reject-tender-offer and announce-restructure-plan are most probably a part of the target company's plan for convincing the raider to increase the offer. These actions are grouped together in an interpretation and the goal of this interpretation is TR-Sweeten-deal.

Situational Representation of a Case  The representation of a case is divided into two components: situational and dynamic. The situational aspect of the case handles the descriptions of the objects involved in the case during case evolution. These object descriptions are stored as States. The initial state of each object is represented by a set of state variables (surface and abstract features) with their associated values. The surface features store the observed descriptions of the objects. The abstract features store the descriptions of the
Figure 2.1: Partial representation of a case
objects which are derived from the surfaces feature using some form of knowledge. The abstract features also have certainty evaluation qualifying the feature value assignments.

**States**  \( X_t \) is the state at time \( t \), where \( X \) is a set of state variables with their associated values. These state variables represent the known values of the features (slots) of the object instances that define the case. The initial state of each object contains all the known values attached to the slots. Following states only contain the incremental changes to the state variables. The value of each state variable at a certain state \( X_t \) is taken from the most recent state-change object that refers to this variable. State changes are temporally linked with other state changes, forming a complete ordering. State changes can be indexed from the event network, by one or more events.

The context in which an event takes place is represented by a state. A state of the world may also be satisfying a pre-condition of an action in some event and that information is also stored with the state. An example of the definition of a state is given in Figure 2.2. In the sequence of states, the state **state-rpp-5** is defined to be after **state-rpp-4** and is followed by **state-rpp-6**. The value-assignments in this state represent changes to the state variables. The event **rpp-e-decrease-tender-offer-PP-01** was enabled by this state. The events, which have **state-rpp-5** in their context are added by the CRL to the **events-at-state** slot which has no value at state definition time.

![Figure 2.2: State Definition](image)

**STATE-RPP-5** is a **STATE**
- **time**: 9/1/85
- **previous-state**: **STATE-RPP-4**
- **next-state**: **STATE-RPP-6**
- **events-at-state**: ()
- **value-assignments**: 
  - ((target-debt-situation :increased)
    - (target-cash-situation :decreased)
    - ((price-per-share ,*Revlon* ) 77.5))
- **enables-events**: (rpp-e-decrease-tender-offer-PP-01)

**2.2 Case Representation Language**

We give a description of the Case Representation Language that we designed in [56]:

To facilitate the organization of case memory, we have designed a A Case Representation Language (CRL). CRL, developed in CLOS [79], is a tool to represent dynamic cases and to provide a mechanism for representing uncertainty in the feature values of the cases.

Using CRL, the information from the cases can be organized around two types of knowledge: conceptual knowledge and episodic knowledge.
1. **Conceptual Knowledge** is the information about the objects, actions, goals which are involved in a case. This knowledge, which for some applications may represent an incomplete domain theory, is used by CARS for case retrieval, case comparison, and solution adaptation.

2. **Episodic Knowledge** is composed of cases. A dynamic case can be thought of a situation/solution pair. The situation consists of the top-level goals and a starting state of the agents; the solution consists of the observable portions of the executions of actions by the agents and their effects on the states of the world. Using CRL these cases can be represented in their actual instantiation without any transformation. The cases are built using the conceptual knowledge.

### 2.2.1 Domain Knowledge Representation

We describe how domain knowledge is acquired in [59]:

The domain knowledge available to the reasoner (and user) can be obtained from expert input, generalization from cases, or extraction from existing KBs. This knowledge is organized into various hierarchies, which are implicitly linked when objects from one hierarchy are used as slot fillers or slot-type specifiers in another.

#### Object Heirarchy

The object and action hierarchies in the M&A domain are described in [2]:

This hierarchy describes the objects of the domain and their relationships. It is a traditional IS-A hierarchy with slots, fillers, and a classical inheritance mechanism.

All the objects that are used in representing the cases are part of this IS-A hierarchy. The objects in the hierarchy are described using slots. [...] In addition to using the object hierarchy for describing objects, the planner uses this hierarchy for analyzing the cases (i.e. for retrieval) and for substituting one object with another similar object (i.e. during plan adaptation).

#### Action Heirarchy

Actions are operations that can alter the states of the objects in the domain. An execution of an action results in some state change. The actions are organized in an IS-A hierarchy that defines an abstraction from special actions with more restricted preconditions and effects to more general actions. [...] 

This hierarchy can be used by the reasoner for deriving a solution. The instances of a particular action class are elements of executed plan actions in the case library. From these instance links, the system can reason about the effects of executing an action. A planning system can use this information (obtained from previous cases) to supplement its knowledge (derived from a weak domain theory) about the effects of actions on state changes.

This hierarchy can also be used by the reasoning system during the solution adaptation phase to perform local search. This process substitutes an action that cannot be performed in the current situation due to resource constraints or failing preconditions with another action that can provide similar effects.
The actions in the hierarchy have implicit links to interpretations of actions in cases. An implicit link between an action and an interpretation is composed of: 1) an instantiation link between the action in hierarchy and its instance in a certain case, and 2) membership link between the action’s instance and an interpretation of actions in that case. These implicit links to interpretations provides a lot of useful information to the planning system, such as actions that need to be generated, and expected actions of other agents. The actions needed are ones that are part of the an interpretation which has implicit link (as defined above) to the planned action in the hierarchy.

The expected actions of other agents in response to a planned action can be generated by using the implicit links between this action in the action hierarchy and interpretations. The implicit links here is composed of: 1) an instantiation link between the action in hierarchy and its instance in a certain case, 2) causal links between the instance of action in a case and actions it had caused in that case, and 3) membership links between the caused action and interpretation of actions in that case. For a planned action, a set of interpretations can be retrieved by using these implicit links. The expected actions, in response to the planned action, is the set of actions that heve membership links to the retrieved set of interpretations.

---

**RPP-A-TO-PP-01 is a Tender Offer**

- **Agent:** *Pantry-Pride*
- **Shares-of-company:** *Revlon*
- **Price-per-share:** 47.5
- **No-of-shares:** 17.95
- **Total-price:** 852.62
- **Payment-unit:** CASH
- **Dollars-per-unit:** 1.0
- **Offer-expires:**

---

**Figure 2.3: Instantiation of Tender Offer action**

An action in one of the events of the hostile takeover attempt of Revlon was Tender Offer by Pantry Pride, and the instantiation of that action in the event is illustrated in Figure 2.3.

**Goal/Plan Hierarchy**

The Goal Hierarchy in the M&A domain [2]:

The Goal hierarchy represents the partial knowledge of the domain theory and provides an initial, albeit incomplete, goal decomposition. Its incompleteness is the reason for resorting to case-based reasoning and mixed reasoning paradigms. The goal hierarchy captures the initial domain knowledge structure and provides a mechanism for expanding it by indexing into each new case at various levels of abstractions (i.e., top level goal of the case, strategies, plan steps, interpretations of single actions, etc.).
The hierarchy is modeled by a tree of And/Or goal nodes. Each node in the hierarchy represents a goal. If the goal of a node can be achieved by achieving the goal of any of its child nodes, then it is an Or node. If the goal of a node can only be achieved by achieving the goals of all its child nodes, then it is an And node. For example, we can observe that one of the raider's goals is acquire-company. This goal is modeled as an Or node since it can be achieved by either acquiring the company with the approval of the company's management (acq-with-management-approval) or without management approval (acq-without-management-approval). The goal-type slot is used to indicate whether the node is an OR node or an AND node. The plans for achieving the goals are indexed by the goals and are stored in the case library. The plan-link slot stores the index to the executed plan. The type of executed plan, which can be an event, an interpretation, or a complete case, is stored in the slot plan-type. The state-vars slot stores the state variables that will change if the goal is achieved.

Each case is linked to one or more nodes in the goal hierarchy. Each link represents the interpretation that the executed action in the case was attempting to achieve a given goal. These links are qualified by a degree of belief indicating the certainty in such an interpretation. For example, the degree of belief in the goal acq-without-management-approval must be higher for the tender-offer action than the buy-stock action. The goals near the top of the hierarchy are very general and are common indices to many cases. As goals are specialized and decomposed into interpretations of events and actions, they provide more specific indices to fragments of the cases.
The underlying assumption used in developing the planner which uses this hierarchy, is that initially the hierarchy will indicate the goals to achieve but may not have all the plans for achieving those goals. On the other hand if all possible plans were captured by this hierarchy, the planning problem could be reformulated in the more traditional generative planning paradigm. Then planning could be based on the selection, refinement and instantiation of plan templates. Our planning system uses this hierarchy like a channel to look for plans in cases. As more cases are added, more (maybe better) plans for achieving the goals in the current situation will be found by the planner. For example, to achieve the goal get-own-shares-back, one case may have the executed plan chunk-buy-back. An addition of another case where the same goal was achieved by a company by swapping shares provides another plan which might be better in some circumstances. The swapping shares plan of the company in the case is composed of actions: return-shares-free (for other company's shares), and get-shares-back-free (for it's own shares). After the addition of the second case the planner retrieves both the plans using the goal hierarchy and then selects the appropriate plan for the current situation.
Chapter 3

Case Retrieval

We describe why case retrieval is so important to the process in [57]:

We believe that case retrieval is of primary importance to the overall effectiveness of any CBR system, for the following reasons:

1. Retrieving the case that will yield the best solution to a new problem ensures the best solution within the system’s capability. This may or may not be the case that matches the new problem the most with respect to superficial (i.e., surface or “raw”) features.

2. Retrieving the case or cases that yield the best solution to a new problem must include some computation of the similarities and differences between the input problem and the retrieved cases. All subsequent case modification uses this computation as a basis.

Methods previously used to determine similarity are discussed in [2]:

Case-based reasoning uses past experiences for doing the task at hand. Therefore, determination of similarity affects all aspects of case-based reasoning. The similarity of salient features identifies the relevant cases, and the similarity of non-salient features of the current and retrieved cases can confirm the relevance. To determine the probability of correctness of an analogy (correctness of relevance of retrieved case), Russell [99] uses the number of total features, salient features and similar features. The dissimilarities of relevant features of a retrieved case can guide the adaptation of the old solution to the new solution. The rest of this sub-section briefly describes some of the work done by other researchers in assessing similarities.

Even though, both CBR and analogical reasoning require retrieving previous instances for reasoning, there are some differences in their similarity assessment. Seifert [102] points out that analogical reasoning typically focuses on inter-domain retrieval, whereas CBR typically performs intra-domain retrieval. Also, exact matches are ideal in CBR, but useless in analogical reasoning. Among other differences, analogical reasoning requires systematic similarity between input and retrieved cases, whereas this requirement may not be needed for CBR as long as the retrieved case can be used in the new problem situation. In this view, similarity is derived from those features (either surface or abstract) of the retrieved cases likely to be useful in the new situation. This set of features changes from situation to situation, so in this sense, the similarity is not systematic.
In CYRUS [81], the assessment of similarities is combined with the indexing process. Cases retrieved during the traversal of the indexing hierarchy are known to be similar to a new case because the cases match on the indexing features. In PARADYME [84, 83], a small subset of best cases is selected from the retrieved cases using preference heuristics.

In general, the MEDIATOR [104] first retrieves multiple cases by following all possible indices. The cases are then ranked according to their similarity to the probe case by a heuristic procedure. This procedure first eliminates all cases in which the most important features (i.e. the disputant's goals) are not identical to the features of the new case. The ranking of the remaining cases is based on how many important features matched the features of the retrieved case.

Multiple similar cases are retrieved by trying to assess similarity along different dimensions. This approach is followed in HYPO [55] for reasoning in the legal domain, and in TACTICAL ASSISTANT [114] for scenario generation in the military planning domain. These dimensions are used as indices and they form discrimination nets. The choice of what features serve as indices is made after knowledge engineering. All cases that match the current case/situation on any of the dimensions are retrieved. In the domain of CBR legal reasoning, certain pre-specifiable features of the input cases are the only features of relevance in finding similar cases, as in HYPO. This constrains the dimensions along which features can usefully be relaxed, and index traversal is done along those dimensions. In HYPO, the cases that had dimensions in support of the reasoner's position and none in support of the opposite position are considered to be most-on-point. The importance of a dimension depends on the context, and in HYPO, the context is characterized by the features of the case and the role a case plays in an argument. In TACTICAL ASSISTANT, cases that do not match on the dimension (situational concept), but are classified nearby, are also retrieved for generating the hypothetical what-if alternatives.

The JUDGE [58] system first "interprets" (determines abstract features) from the "actions and results" (surface features) of the "crime" (case). The results of interpretations are used as indices for finding similar cases. Determination of salient features is done on a case by case basis by using the causal structures built while interpreting each case.

The retrieval of a story in the CreANIMate system is based on its educational objective. When it is retrieving a story to present as an explanation of an animal morphology under consideration, it uses either the feature/function index or function/behavior index. The former index is used to retrieve stories that exemplify the relation between certain physical features (i.e., long legs) and functions performed (i.e., run fast) by animals while the later index is used to retrieve stories that exemplify the relation between the functions performed (i.e., run fast) and a high level survival behavior (i.e., purse-prey). To retrieve a story that is related at an abstract level to the explanation of animal morphology under consideration, it uses the abstraction information encoded along with the indexing information in each case. For example, the abstraction information on a story that has the index run-fast to pursue-prey may indicate that this story can be abstracted up to the abstract level of move-fast to hunt. To retrieve a story that would give an expectation violation of the animal morphology under consideration, it uses the rules that indicate the expectations a student might have for certain animal morphologies. The retrieval is performed by searching the hierarchy of rules to see if one applies, and in case it does the story indexed by the rule is retrieved. The retrieval of cases (stories) in CreANIMate is different from other systems in the sense that it
does not use the contents of the story/case but uses the information encoded with the story for determining which case (story) should be retrieved.

Another system, Broadway [105], also includes in the case the information that will be useful in retrieving the case. In this approach knowledge sources are created from the cases. These knowledge sources have preconditions that are local to specific type of knowledge sources. If these preconditions become true for a certain input case then this knowledge source will post the case as relevant to current problem solving. This approach provides for handling special considerations for similarity determinations depending on the cases, because the similarity determination is based on the evaluations of the pre-conditions of their knowledge sources.

Another interesting aspect of similarity determination is representation and reuse of similarity [91]. This aspect is not addressed very much in the research on similarity assessment but it may help in complex domains. The concepts of preconditions used in the Broadway system may be useful for representation and reuse of similarity.

In Redmond’s work [94], cases are represented as snippets. The retrieval of a case translates into retrieval/access of snippets. At each step of diagnosis, the next snippet is accessed either sequentially by following links between snippets of the same case or directly through retrieval which uses the current situation, snippet’s goals and context. The direct retrieval of snippets is done using the goal as an index. The selection of a snippet from the retrieved snippets is accomplished using a weighted similarity metric for matching. The match is done on all the features in the internal and global context of snippets. The weights on the feature's importance may be adjusted using the success or failure of prediction during learning.

Case retrieval can be broadly categorized into three types: those which use pre-determined indexing techniques (i.e., CreANIMate) for fast retrieval, those that group cases which share general features (i.e., CYRUS), those that base their retrieval on the contents of a case. Our approach falls in the third category.

In this third category in general, case retrieval and similarity assessment are differentiated. Abstract features (i.e., dimensions in HYPO, interpretations of features in JUDGE) are derived from the raw features of the cases and are used for similarity assessment. After they derive the needed abstract features they basically use these features as indices for retrieving cases. They do not do a partial matching on these features, so they do not really address the problem of aggregating the partial matches. Also none of the research on case retrieval, other than ours [30], addresses the problem of how to assess similarity when there is uncertainty associated with the case features. No other case-base reasoning system reasons with dynamic cases therefore none of these systems try to find similarity between sequences of events.

3.1 Analysis of Cases

Case analysis is a critical step in the CBR process, because it is this analysis which determines the salient features of a case, which in turn determine the suitability of that case as a potential matching case. The procedure for the analysis of cases is given in [2]:

Domain specific knowledge is used for analyzing cases to derive abstract features. These features are assigned a value and a degree of certainty. Values for features (abstract or
surface) can be raw data or lexical terms (linguistic values representing fuzzy intervals [120]) chosen from feature value term-sets provided in the planning system. The degree of certainty represents the extent to which the abstract features can be inferred from the surface features.

The companies in a case are analyzed along six categories that are financial through relation-with-other. One or more abstract features are derived for each category. Figure 3.1 shows all the abstract features and their categories. To analyze a case along a certain category, only the abstract features for that category have to be derived. For example, to analyze the financial situation, only short-term-fc, long-term-fc, coverage, and profitability abstract features have to be derived. As discussed earlier, both plausible rules and conceptual knowledge is used for deriving abstract features. Abstract features for the relation-with-other category are derived using CRL conceptual knowledge. Abstract features for all the other categories are derived using the PRIMO plausible rules. These rules are organized into various rule classes and the rule classes for each abstract feature are also shown in the figure. In this section we will discuss in detail how one abstract feature short-term-fc is derived using a PRIMO rule.

The derivation of the short-term-fc abstract feature is determined using five PRIMO plausible rules which are denoted by rectangles in the figure. One of these rules, Acid-Ratio-St-Fc, is illustrated in Figure 3.2.

The rule in Figure 3.2 consists of a rule name, rule class, instantiation class, object variables, documentation, context, antecedent, consequent, and rule strength. These rule components are used for 1) rule base design, 2) rule instantiation, 3) control of inference, and 4) rule evaluation.

1) Rule Base Design: Rule name and rule class are used to identify the rule and structure the rule base for the purposes of efficiency in inference and ease of debugging and knowledge engineering.

2) Rule Instantiation: Rules are written with object variables scoped by an implicit universal quantifier. While rule classes are design partitions of the rule base, Instantiation classes are instantiation partitions of the same rule base, i.e., they define the subsets of rules to be jointly instantiated when a new instance of an object occurs. Also, object variables are instantiated with the corresponding slot values of the new instance. In our example, they are ?company and ?industry-ratios.

3) Inference Control: The Context is a pre-condition that must be satisfied before the antecedent of the rule is evaluated. Typically a context is a conjunction of predicates on object-level variables (i.e., domain variables) or meta-level variables (i.e., processing resources and requirements). In our example, they perform a type checking on the value of the predicates used in the antecedent (to guarantee that all numeric values are available).

4) Rule Evaluation: The Antecedent is a conjunction of (possibly) fuzzy predicates on object-level variables. The conjunction is implemented using T-norms [34], which are described below. The result of the antecedent is the degree to which the conjunct of predicates is satisfied. The output of the antecedent, in conjunction with the Rule Strength, is used to determine the truth value of the Rule Conclusion. In our example we have one predicate acid-ratio-pred, which computes the acid ratio of the company as:

\[
\text{AcidRatio} = \frac{\text{CurrentAssets} - \text{Inventory}}{\text{CurrentLiabilities}}
\]
Figure 3.1: Analysis of a company
and normalizes it with respect to the industry average acid ratio. The mapping illustrated in Figure 3.3 is then used to select the term that best describes the short term financial condition of the company, given the acid ratio average of its industry sector.

In our implementation, the intervals used in the mapping are actually fuzzy intervals. Therefore, the membership value of the acid ratio percentage is computed for each term in the termset. The term with the highest membership value is selected. The corresponding membership value describes the degree of confidence of this linguistic value assignment.

### 3.2 Feature Value Comparisons

By analyzing both the probe and the retrieved case, a linguistic value's label is obtained for each of the abstract features. Each linguistic value's label has a meaning defined in its term set. For example, the labels and their semantics in the financial condition termset are given in Figure 3.4.

In the second column of Figure 3.4, a parametric representation is used to describe the membership distribution of each term, \( N_i \). Using this representation, a fuzzy set of a universe of discourse \( U \) can be described as a four-tuple: \( (a, b, \alpha, \beta) \). The universe \( U \) is a unit
<table>
<thead>
<tr>
<th>Acid Ratio Percentage Interval</th>
<th>Linguistic Value’s Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0,60]</td>
<td><em>VERY-WEAK</em></td>
</tr>
<tr>
<td>[60,80]</td>
<td><em>WEAK</em></td>
</tr>
<tr>
<td>[80,90]</td>
<td><em>BELOW-AVERAGE</em></td>
</tr>
<tr>
<td>[90,115]</td>
<td><em>AVERAGE</em></td>
</tr>
<tr>
<td>[115,140]</td>
<td><em>ABOVE-AVERAGE</em></td>
</tr>
<tr>
<td>[140,170]</td>
<td><em>STRONG</em></td>
</tr>
<tr>
<td>[170, ∞)</td>
<td><em>VERY-STRONG</em></td>
</tr>
</tbody>
</table>

Figure 3.3: Mapping of Percentage Acid Ratio to Terms Labels

<table>
<thead>
<tr>
<th>Term Label</th>
<th>Term Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>VERY-WEAK</em></td>
<td>(0 130 0 20)</td>
</tr>
<tr>
<td><em>WEAK</em></td>
<td>(170 270 20 30)</td>
</tr>
<tr>
<td><em>BELOW-AVERAGE</em></td>
<td>(310 410 30 30)</td>
</tr>
<tr>
<td><em>AVERAGE</em></td>
<td>(450 550 30 30)</td>
</tr>
<tr>
<td><em>ABOVE-AVERAGE</em></td>
<td>(590 690 30 30)</td>
</tr>
<tr>
<td><em>STRONG</em></td>
<td>(730 830 30 20)</td>
</tr>
<tr>
<td><em>VERY-STRONG</em></td>
<td>(870 1000 20 0 )</td>
</tr>
</tbody>
</table>

Figure 3.4: Linguistic values for Financial condition termset

interval (represented by an integer representation on the scale from 0 to 1000). The first two parameters \((a, b)\) indicate the interval of the universe of discourse in which the membership value is 1.0; the third and fourth parameters \((\alpha, \beta)\) indicate the left and right width of the distribution. Linear functions are used to define the slopes. Let \(\mu_{N_i}(x) : X \rightarrow [0,1]\) be the membership function of the fuzzy set \(N_i\), as illustrated in Figure 3.5.

The fuzzy set \(N_i\) can be represented as a four-tuple \((a_i, b_i, \alpha_i, \beta_i)\) where:

\[
\mu_{N_i}(x) = \begin{cases} 
0 & \text{if } x < (a_i - \alpha_i) \\
\frac{1}{\alpha_i}(x - a_i + \alpha_i) & \text{if } x \in [(a_i - \alpha_i), a_i] \\
1 & \text{if } x \in [a_i, b_i] \\
\frac{1}{\beta_i}(b_i + \beta_i - x) & \text{if } x \in [b_i, (b_i + \beta_i)] \\
0 & \text{if } x > (b_i + \beta_i)
\end{cases}
\]

The membership distribution described by the above equation is illustrated in Figure 3.5.

Having established the meaning of the labels used to define each abstract feature value, we will now discuss how the similarity measure for each abstract feature is determined. This is done by executing a two step procedure.

The first step, referred to as degree of matching determination, consists of computing the closeness of two linguistic values based on their semantics. Initially, the distance between
the fuzzy set representations of the corresponding values is computed. For example, let us assume that the abstract feature Target-Short-Term-Fc-Sim has the value *STRONG* in the probe case and *VERY STRONG* in the retrieved case. The distance between the two corresponding fuzzy sets is computed as the absolute value of their difference. This is done using fuzzy arithmetic operations that are closed under the four-tuple parametric representation [41, 33, 31]. Specifically, given two fuzzy numbers \( X = (a, b, \alpha, \beta) \) and \( Y = (c, d, \gamma, \delta) \) we can define the difference

\[
X - Y = (a - c, b - d, \alpha + \delta, \beta + \gamma).
\]

In this example, the difference between *VERY-STRONG* and *STRONG* is \((40, 270, 40, 30)\). This distance is then transformed into a degree of matching by taking the complement with respect to the unit interval. Using the same formula for the difference, by representing the unit as \((1000, 1000, 0, 0)\), the degree of matching \(1 - |X - Y| = (730, 960, 30, 40)\).

The second step, referred to as linguistic approximation, consists of selecting a label (chosen from one of the similarity term-sets provided) whose meaning is the closest to that of the computed degree of matching. This semantic closeness is evaluated by a measure of set inclusion [50]:

\[
\frac{|P \cap D|}{|D|}
\]

where \(P\) is the similarity term and \(D\) is the result of complementing the set-distance. This measure, representing the degree of matching between the reference \((P)\) and the data \((D)\), is used as an associated certainty value for the label. A detailed study of measures of inclusions is given in [42] (page 23-24).

A simple example of a seven term similarity termset is given in Figure 3.6.

The degree of matching between *VERY-STRONG* and *STRONG*, as computed in the last example, is a fuzzy number \((730, 960, 30, 40)\). By using the termset described in Figure 3.6, one can see that the term with the closest meaning \((730, 830, 30, 20)\) is *ALMOST-COMPLETE-MATCH*. The degree of confidence in this label selection is

\[
\frac{|(730, 830, 30, 20) \cap (730, 960, 30, 40)|}{|(730, 960, 30, 40)|} = \frac{125}{265} = 0.47.
\]
<table>
<thead>
<tr>
<th>Term Label</th>
<th>Term Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>NO-MATCH</em></td>
<td>(0 130 0 20)</td>
</tr>
<tr>
<td><em>ALMOST-NO-MATCH</em></td>
<td>(170 270 20 30)</td>
</tr>
<tr>
<td><em>LESS-THAN-PARTIAL-MATCH</em></td>
<td>(310 410 30 30)</td>
</tr>
<tr>
<td><em>PARTIAL-MATCH</em></td>
<td>(450 550 30 30)</td>
</tr>
<tr>
<td><em>MORE-THAN-PARTIAL-MATCH</em></td>
<td>(590 690 30 30)</td>
</tr>
<tr>
<td><em>ALMOST-COMPLETE-MATCH</em></td>
<td>(730 830 30 20)</td>
</tr>
<tr>
<td><em>COMPLETE-MATCH</em></td>
<td>(870 1000 20 0)</td>
</tr>
</tbody>
</table>

Figure 3.6: Termset For Partial Matching of Abstract Features

From the same Figure 3.6 one can see that the term *COMPLETE-MATCH*, with its meaning described by (870, 1000, 20, 0), has a degree of confidence of

\[
\frac{|(870, 1000, 20, 0) \cap (730, 960, 30, 40)|}{|(730, 960, 30, 40)|} = \frac{120}{265} = 0.45.
\]

Therefore the term *ALMOST-COMPLETE-MATCH* is selected as the value for the similarity measure for the abstract feature Target-Short-Term-FC-sim.

Multiple similarity term sets are used to have different “views” of similarity (e.g., the lenient similarity term set has wide fuzzy intervals for the labels representing high similarity and narrower intervals for those representing low similarity. The opposite is true for the strict term set.)

### 3.3 Combination of Similarities

The similarity measures can be aggregated or chained (using the transitivity of similarity) according to well-defined operators called triangular norms. Triangular norms (T-norms) are the most general families of binary functions that satisfy the requirements of the conjunction operators. T-norms are two-place functions from \([0,1] \times [0,1]\) to \([0,1]\) that are monotonic, commutative and associative. Their corresponding boundary conditions, i.e., the evaluation of the T-norms at the extremes of the \([0,1]\) interval, satisfy the truth tables of the logical AND operator [51], [52], [34]. Five uncertainty calculi based on the following five T-norms are used:
\[ T_1(a, b) = \max(0, a + b - 1) \]
\[ T_{1.5}(a, b) = \begin{cases} (a^{0.5} + b^{0.5} - 1)^2 & \text{if } (a^{0.5} + b^{0.5}) \geq 1 \\ 0 & \text{otherwise} \end{cases} \]
\[ T_2(a, b) = ab \]
\[ T_{2.5}(a, b) = (a^{-1} + b^{-1} - 1)^{-1} \]
\[ T_3(a, b) = \min(a, b) \]

Their corresponding DeMorgan dual T-conorms, denoted by \( S_i(a, b) \), are defined as:
\[ S_i(a, b) = 1 - T_i(1 - a, 1 - b) \]

These five calculi provide the user with an ability to choose the desired uncertainty calculus starting from the most conservative \( (T_1) \) to the most liberal \( (T_3) \).

The use of T-norms in aggregating and chaining certainty intervals during the extraction of abstract features is extended in CARS to the aggregation of similarity measures. This mechanism aggregates similarities by taking as input a list of similarities to be combined, their associated uncertainties, and optional weights indicating the importance of the feature in the aggregation. This mechanism is based on three aggregation operators: T-norms, T-conorms, and Linear combinations.

T-norms are used to discount low similarities
T-conorms are used to enhance high similarities
Linear combinations are used to average remaining similarities.

First the low and high values of similarity are aggregated; (weighted) low values are aggregated using the minimum operator (with the option of using other T-norms), while (weighted) high values are aggregated using the maximum operator (with the option of using other T-conorms). The result of these partial aggregations (multiplied by the cardinality of the aggregated values) are averaged with the intermediate values of similarity.\(^1\)

First, this process normalizes the similarity values of various abstract features according to their relevance weights. Then the process penalizes bad matches and rewards good ones. Finally, the process considers tradeoffs by averaging the remaining intermediate values with the previous results. A detailed study of aggregating operators is given in [43].

\(^1\) Let \( \bar{X} \) and \( \bar{W} \) be two \( n \)th dimensional vectors with elements in \([0,1]\). \( x_i \in \bar{X} \) represents the similarity value of the \( i \)th abstract feature, while \( w_i \in \bar{W} \) is its corresponding relevance weight. The weighted minimum \( \text{WMIN}(\bar{W}, \bar{X}) \) is defined as
\[
\text{WMIN}(\bar{W}, \bar{X}) = \bigwedge_{i=1}^{n} (w_i - x_i) = \bigwedge_{i=1}^{n} \max(1 - w_i, x_i)
\]
In Figure 3.7, two levels of aggregation of similarity measures can be observed: All the relation abstract features similarities (from raider-target-location-relation-sim to raider-target-business-relation-sim) are aggregated to determine the raider-target-relation-similarity between the probe and the case. All the raiders abstract feature similarities (from Raider-short-term-FC-Sim to Raider-Performance-As-Raider-Sim) are aggregated to determine the Raider-Raider-Similarity between the probe and the case. Similarly, the target abstract feature similarities are aggregated to derive the Target-Target-Similarity. Finally, Target-Target-Similarity, Raider-Raider-Similarity and Raider-Target-Relation-Similarity are combined to derive the case Phase-1-Similarity.

During the aggregation process of similarity, we do not account for slots with no values (its certainty is denoted by the extended certainty bar, representing complete ignorance) in determining the similarity. But in aggregating the certainty of similarity, we do use the certainty of that slot (which is complete ignorance). This reduces the certainty in the overall result which should be the case when there are missing values.

Most of the target abstract feature similarities range from *COMPLETE-MATCH* to *LESS-THAN-PARTIAL-MATCH* with varying degrees of certainties. Their aggregation is performed using T-conorms and linear combination operators and it results in an *MORE-THAN-PARTIAL-MATCH*. The last aggregation to obtain Phase-1-Similarity returns a similarity of *MORE-THAN-PARTIAL-MATCH*, as shown in Figure 3.7.

The similarity between two cases can be computed by aggregating the five phase-similarities in an analogous fashion. It is straightforward to customize this final aggregation to reflect different goals of the retriever. Let us recall the definition of the five phases: (1) Initial Condition, (2) Pre-tender, (3) Tender-negotiation, (4) Outcome and (5) Long-term results.

For instance, by only aggregating phases 1, 2, and 4 one can stress the need to find successful cases with similar initial and pre-tender conditions. The result can give the range of tender-negotiations (plans/counter-plans) which are applicable to the current situation.

Alternatively, by only aggregating phases 2, 3 and 4 one can observe the range of macro-economic conditions to ascertain raider/target financial assessments, for which a particular (pre-tender and post-tender) plan was successful.

Similarly, the weighted maximum \( WMAX(\bar{W}, \bar{X}) \) is defined as

\[
WMAX(\bar{W}, \bar{X}) = \sqrt[n]{\min_{i=1}^{n}(w_i, x_i)}
\]

In our system we use linguistic values (with fuzzy numbers semantics) to represent similarity and weights. Therefore we have extended the above operations to fuzzy numbers in \([0,1]\) using the four parameter representations and the formulae in reference [31].
Figure 3.7: Aggregation of Similarities
We give a general overview of the planning process [2]:

Planning can be defined as the problem of deciding the sequence of actions that will transform the given initial state of the world into the desired goal state. Along with the problem of deciding the sequence of actions, the planning research in the Artificial Intelligence (AI) community also addresses the issue of how to make sure that the goal state is reached. This second planning task requires the planner to keep track of the different world states and to modify (or refine) the plans if desired. This requirement is the basic difference between AI and non-AI planning systems. Typical operations research [109] tools, like CPM and PERT, do not represent the causal relationships between actions and cannot reason about the effects of their actions nor can they revise their plans.

Planning can be divided into two categories: strategic and tactical planning. Strategic planning is concerned with producing the sequence of actions for the long term. The strategic planning system, using its knowledge about actions and their effects, chooses between possible courses of actions. Typically, the sequences of actions are not completely ordered and are at a higher level of abstraction. Tactical planning (reactive planning) involves a constant feedback from the state of the world in which planning is being done. It is concerned with generating actions for the short term in the context of the current world state.

Plans for reaching a goal state from a given initial world state can be developed in two ways: a whole plan can be generated from scratch or a previous plan can be modified. In Generative planning, the planner decides which action should be taken first by taking into account constraints, the initial situation, and the final goal state. From its knowledge about the effects of actions, its projects a new state that will be reached after the action(s) is performed and then selects the next action. This process is done recursively until the entire course of actions that will take the planner to the goal state is generated. In Case-based planning, the sequence of actions (the stored plan) which solved a previous similar problem is retrieved. The previous plan is modified, so that the actions which do not contribute to reaching the goal state are dropped/replaced. New actions may be added to overcome some situation that was not present in the previous problem.

And plan development:

In case-based planning (CBP), the existing plans are used in planning for a new situation. The CBP systems emphasize how to modify/adapt a retrieved plan rather than producing just one answer as in CBR (i.e., jail sentence in JUDGE). The cases in the case memory of
a CBP system consists of situational context (events, constraints, and goals) and a solution (the plan executed at that time). The task is to retrieve the case with the most similar situational context and to adapt the solution from that case for use in the situational context of the probe. Case-based plan representation can be compared to both the state space representation and the action ordering representation of generative planning. Initially, the retrieved relevant plan has goals/subgoals, states of the world, and the actions that were carried out to obtain those states. This state space representation of previous plans is then adapted using the domain knowledge, other cases, current goals, and context to obtain the new plan (list of action).

Case-based planning differs from the more traditional generative planning in many aspects. Generative planners build the plans in many micro steps and while building, the planner has to ensure that the plan will achieve the goal state from the given initial state, constraints, and resources. The case-based planner retrieves the plans in one step and then adapts them to ensure that they will achieve the goal. The major system resource used by the generative planner is computer time needed for generating a plan while the case-based planner uses the storage space of the system for storing previous cases. The generative planner requires a good domain theory so that it can extrapolate the effects of actions that it has generated for achieving the goal. The case-based planner requires a wealth of previous cases so that it can find relevant cases close to the current situation.

One of the earliest and best known examples of a case based planner is the CHEF system, built by Kristian Hammond [72]. The CHEF program addresses the problem of planning in the cooking domain. It generates new plans (recipes) by adapting the sequence of actions from similar past plans (recipes). The input to CHEF is a list of goals (such as a hot stir fry dish with chicken and broccoli) that have to be satisfied. The result of planning by CHEF is a plan that satisfies these goals. If part of a plan fails, CHEF repairs the plan and an index to the repair is added to memory to avoid repeating that planning failure. CHEF does not have any interpretation of input goals for the retrieval of plans, but it does exhibit complex plan adaptation and learning capabilities.

CHEF retrieves similar cases based only on goal similarity. It does not address how to retrieve a recipe which is similar to an incomplete recipe (where the incomplete recipe may have goals along with a partial description of actions already taken). Therefore, when a plan fails during execution, due to failed preconditions or objectionable results, CHEF stores the failure, but must begin from scratch to rebuild the plan (recipe). One major requirement of the problem domain which we are addressing is the ability to continue planning from any point in plan execution, while maintaining consistency with previous actions. This is the result of having to deal with other (possibly antagonistic) agents changing the world state. This is one of the major differences between the CHEF system and our work. Our system can retrieve a plan which will be most suitable for continuation of a partially executed plan.

If planning involves multiple agents then the planner has to resolve the conflict of goals of these agents while developing plans. PERSUADER [108] first presents a plan and if that plan is rejected due to conflicting goals of the agents, it follows two options to resolve this conflict: it generates persuasive arguments to convince the rejecting agent or it modifies/repairs the plan to make it more acceptable. To persuade a rejecting agent it uses a goal hierarchy. It takes the goal of the rejecting agent and uses the goal hierarchy to find some goal with higher importance which will be effected if the agent tries to achieve the conflicting goal. To
modify/repair the plan, it uses the reason of rejection (explanation) given by the rejecting agent for repair. It then uses explanation-based similarity retrieval to retrieve a plan that fixes that problem. This system uses the goal hierarchy and the knowledge encoded in it to persuade the agents. Our system can also use the goal hierarchy, but it also uses previous cases to see how others were persuaded in past cases and from those examples it determines a new action that should persuade. This approach then does not depend on encoding the domain knowledge but relies on and leverages past cases.

Adaptive planning is somewhere in between Case-Based Planning and Generative Planning. It attempts to mix old specific plans with general plans while developing a plan for a current situation. PLEXUS [54] is an adaptive planner that successfully adapts a specific plan from an old situation to work for the current situation. The flexible utilization of the old plans is done by using: 1) background knowledge associated with an old plan for situation matching, 2) a specific plan for an old situation, and 3) treating the failing steps of the old plan by representing the categories of actions that have to be achieved. The background knowledge associated with an old plan is determined by the old plan's position in a knowledge network. The network includes: taxonomic structure for property inheritance and reasoning about categories, partonomic structure (step-substep hierarchy) for refitting actions, and causal knowledge that includes relations such as purpose and reason. Each of the steps (substeps) in the old plan have appropriateness conditions like precondition, outcome, and goal associated with them. A situation difference occurs between the old plan and current situation if one of the appropriate conditions fails or the steps in that plan are out of order. To correct the situation difference, PLEXUS treats the failing plan as a category of action and uses the background knowledge for finding a substitution. This is accomplished by first abstracting until a category of plans common between the two situations is found, and then specializing until an alternate course of action appropriate for the current situation is found. It utilizes various rules for abstraction and specialization to ensure that efficient and correct substitution is done. To handle the step out of order situation difference, it uses relations like reason to look ahead at the effect of those steps in the previous plan.

More recent work in planning by Hammond et al., [73] [65] addresses the issues of opportunism and flexible plan use in the areas of reactive planning and strategic/tactical planning. In RUNNER, the observation of particular values of environmental features (state) triggers the activation of a goal(s), which is used to index into memory to retrieve an existing plan for satisfying the goal(s). The retrieved plan is used to give permission to sub-plans/actions to take place. Opportunity for an action depends on the observation of particular features in the environment. An action must have both permission and opportunity to be executed. In summary, the guidance on permissible actions comes top-down from the goals, and recognition of opportunity comes bottom-up from the state. The action(s) which lies on their intersection is taken.

All case-based planning systems retrieve a plan for the goal they are trying to achieve. We first retrieve the most similar case in which actions were done by various agents. From this we identify a plan. During adaptation of a plan we rely on past cases and the explanations that an expert has given, while the current planners rely on the informations encoded in their plan modifiers. We also use the uncertainty in cases while identifying a plan and adapting it.
4.1 Plan Extraction

Plan extraction in the M&A domain [2]:

The cases in our planning system are observed episodes of takeover battles. We do not make the assumption like other case-based planners (CBP) that the retrieved case is a specific plan of a past situation. Since our case is an actual episode of what happened in the past, we have to analyze this episode and extract a plan of the agent whose executed actions are part of this case. This step can be added as a pre-processing step to any of the existing CBP to enhance their capability so that they will be able to use cases that are not just specific plans.

4.1.1 Representation of plan

The plan of an agent consists of strategies and planned actions. The strategy encapsulates the information about each sub-plan of the planner. This information consists of the subgoal of this sub-plan, the higher level goal whose achievement depends on successful completion of this strategy, the executed plan in the stored case from which this sub-plan was extracted, the totally ordered set of steps that have to be executed for this sub-plan, and the time consideration (i.e., phases) in which this sub-plan has to be executed. The planned action\(^1\) encapsulates the local information about each step in the plan. This information consists of the goal of this step, the actual action \(a_i\) that will be executed to achieve the goal, the strategy of which this planned action is a part of, the action from which this step was identified, the actions that have to be performed with and/or before the execution of the action \(a_i\), and the stage of planning through which this planned action has passed.

The lower right box in Figure 4.1 on page 34 shows the information for the strategy initial-open-purchase-1024. The goal of the strategy is to buy the stocks of the target company in open market before their prices go up and this goal is represented as initial-open-purchase. The planned step for this strategy is the planned-action buy-stock-1027. This plan has to be executed at the time when it is not know for sure that the target company is a potential target for a takeover. This information is stored in the start-phase and end-phase slots of the strategy object. This strategy has to be executed in phase-2 of the takeover.

The goal get-share-holders-to-sell of this planned action is a subgoal of the raider's goal to acquire-in-open-market the stocks of the target company. This goal can be achieved by executing the actual action tender-offer-1028 which was identified from action rpp-a-to-pp-01 in the stored case. The plan-stage of this planned action is structural which represents that this action has been structurally adapted to current situation.

4.1.2 Identification of a plan

The identification of a plan to achieve a goal \(g_i\) by an agent in the probe case is done using the role \(r_i\) of this agent, the goal hierarchy, and the information in the retrieved case. This identification of a plan is done in three steps: the identification of all possible events for the

\(^1\)Sometimes we use the word planned step instead of planned action for readability.
plan, the compression of the plan by removing unnecessary actions, and the linking of the parts of this plan to the hierarchical expansion of the goal \( g_i \).

First, the identification of all possible events for the plan is done using the role \( r_i \). All events in the retrieved case, that have actions executed by an agent whose role is the same as \( r_i \), are identified as possible events for the plan. All the identified events are added to the set \( P_a \) of events. These identified events include the actions planned by the agent for achieving the goals and also the actions that had to be executed for other reasons such as countering the actions of the opponent.

Then the compression of the plan is done using the contextual information, encoded as links, of the events in the set \( P_a \). This step removes the enabling events, compensatory events and reactory events. The enabling events of an event \( e_i \) are those events whose execution caused a new state which is the enabling state of the event \( e_i \). The enabling events of an event \( e_i \) are determined by using the enable-link to get the enabling state and then using causal-links to get the events which caused the enabling state. All those events that are enabling events for other events in \( P_a \) and do not have a goal which is a descendent of goal \( g_i \) are removed from the set. These events are removed because they were executed to enable a planned action and they did not contribute to the achievement of the goal. The compensatory events of an event \( e_i \) are those events that were executed to compensate for the lack of expected impact of the executed action of event \( e_i \). The compensatory events of an event \( e_i \) are determined by using the membership-links to get the compensation interpretation and then using parts-of links to get all the events grouped by the compensation interpretation. All the events in this group except the event \( e_i \) are compensatory events. Also as a convention this event \( e_i \) is always the first event in a compensation interpretation. All the compensatory events are removed from the set \( P_a \). The need for these compensatory events would only be known after the execution of the action, therefore they need not be included in the strategic plan. The reactory events are those events that were caused by the execution of an action by the opponent agent. The reactory events of an event \( e_i \) are those events that are linked to event \( e_i \) by a causal link. These events are also removed from the set \( P_a \). Our approach to plan compression relies on the contextual information of the events in the cases unlike plan compression in CHEF [72] which uses the information about the actions encoded in the knowledge base.

Finally, the linking of the parts of the compressed plan to the hierarchical expansion of the top level goal \( g_i \) is done in a bottom up way. The remaining events in \( P_a \) are linked to goals in the hierarchy. These events are grouped into sub-plans which are linked to higher level goals in the hierarchy, and then the sub-plans are recursively grouped and linked to the next higher level goals till we reach the top level goal. This grouping of events into sub-plans can be done either by using plan-steps interpretations (if there are any) or by using the goal hierarchy. The events that are part of plan step interpretation are grouped into a sub-plan and the goal of this sub-plan is the goal of the interpretation. The rest of the events in \( P_a \), which can not be grouped using interpretations, are grouped using the goal hierarchy. The events, whose goals have the same parent goal node \( \text{par}_g \), are grouped into a sub-plan. The goal of this sub-plan is the parent goal \( \text{par}_g \). This grouping also checks whether the goal node \( \text{par}_g \) is an AND node and in that case if each of the sub-goals of \( \text{par}_g \) have a plan in the newly formed group. In case some sub-goals do not have a plan, these sub-goals are added to the list of unresolved-goals. For each of the remaining ungrouped events, which do not have
Figure 4.1: Plan extracted for the Raider

a parent goal common with other events in set \( P_i \), a sub-plan consisting of only that event is made. The goal of this new sub-plan is the parent goal of the event’s goal. If this goal is an AND goal then all its subgoals except the goal of the event in the sub-plan are added to the list of unresolved-goals. These identified sub-plans are recursively grouped together using the goal hierarchy until we reach the top goal \( g_i \).

At the end of this final step of the extraction of the plan, we have identified the parts of the plan from the retrieved case which can be used for goals that contribute to the achievement of the top level goal of the agent. This identified plan is then represented using strategies and planned actions. A planned action is made for each event in the extracted plan. The example of a planned action was given earlier. This planned action can also have information about other planned actions (not shown in the example). For example, if the event for which a planned action is made is a member of the two-agent interpretation, then slot with-action of the planned action will indicate the action that needs to be executed with this planned action along with the possible agent. This action is of the same type as the action of the other event in the two-agent interpretation. The possible agent is the one which has the same role in the current situation as the role of the agent of the action in the other event. For example, a transaction type planned action will have to be planned to be
executed with an action by another agent.

After completion of plan extraction, we have a list of unresolved-goals that the goal hierarchy indicates are necessary but that did not have plans in the retrieved case. In our planning system we give the user the option of identifying the goals to be resolved. As a default the system resolves all the goals. The resolution of these goals is discussed in the next section.

The plan extracted for the raider in the probe case is given in Figure 4.1. This plan is extracted for achieving the top level goal acq-without-management-approval of the raider. It consists of three sub-plans represented by strategies initial-open-purchase-1024, acquire-in-open-market-1025, and force-management-into-selling-1026. The goals of these subplans are the sub-goals of the top level goals. Therefore, these sub-plans are not further grouped. The goal node of acq-without-management-approval is of type OR and its sub-goals have plans identified. Therefore, after extraction of this plan there are no unresolved goals. The planned actions for each strategy are linked to their strategy as shown in the figure.

4.1.3 Identification of resources and constraints

After the extraction of a suitable plan from the retrieved case and the identification of the goals that may have to be resolved, the planner sets up its view of the world. This view includes resources of the planner, its constraints, and what it knows about the opponent agent. The initial information in these views are the state, the phase of the takeover, the resources of the planner, and the price range in which the agent can operate.

In a hostile battle this price range indicates the minimum and maximum price-per-share of the target company in which the battle can take place. For the raider, the lower bound of this price indicates the minimum price he can start with and the higher bound indicates the maximum price he can offer. He will have to abort his takeover attempt if the trading price per share of the target company goes over the maximum or some other company offers more than the maximum. For the target, the lower bound indicates that he has to reject any offer below it and the upper bound indicates that he has to accept any offer above it. These ranges are local to each view. A raider view may have a different range from the target view. A study of various methods of determining the price of companies is given in [86] (pages 38-79). We use two of these methods to determine the price range.

The two methods we use for determining the price for the shares of a company rely on two types of knowledge. One method requires the experiential knowledge of a person, that is what he has seen in the past as the price paid for the shares of a company which was in a situation similar to the target company. This method fits well with our Case-Based reasoning technique. The second method requires general knowledge for determining the quantitative value of the target company to the raider. This knowledge is stored as rules.

We use both methods to determine a low and high premium that should be acceptable for the current price of the target company's shares. This premium is in the form of a percentage of the company's share price. For the first method, the most similar target company is the target company in the retrieved case. We analyze all the actions of type shares-handling-actions in the retrieved case to get the highest and lowest premium offered by any agent for the shares of the target company. The premium offered by an agent in each of these actions
Figure 4.2: The factors for Quantitative value

is a function of the price-per-share offered by the agent and the price-per-share of the target company shares in the state in which the action was executed. For the second method, the knowledge of the quantitative value is represented as rules whose rule classes are shown in Figure 4.2. Each rule class determines the quantitative value based on that factor. For example, the operational factor determines how much of value is the target company to the raider based on the operations of the target. This includes such things as do the operations of the target give synergy to the operations of the raider in cases when the target makes the same products as the raider or when he has the same type of distribution channels, does the target operate in a market which is hard to enter, and does the target have a good market share. The evaluation of the above rules will indicate the quantitative value of the operational factor. The sum of the values of all four factors will give the quantitative value of the target company to the raider company. This quantitative value is also represented as a percentage. These two methods give us three premium values in the form of percentages for the share price of the target company. After determining the three premium, the low price of the price range is determined by changing the current price-per-share of the target company's shares to the minimum percentage in the three premiums. The high price of the price range is determined by changing the current price-per-share of the target company's shares to the maximum percentage in the three premiums.

The use of these two methods shows another example of combined reasoning performed by the system to solve a subproblem during planning.

Figure 4.1 shows the initial view of the raider. The current state of the raider is state-kpm-phase1 and the current phase is phase-1. This phase reflects that no action has been
taken by the raider yet. The funds available to the raider in this state are $2418.0 million. The price range for shares in which the raider should attempt to takeover the target company is $60.12 and $87.174 for each share of the target company.

4.2 Planning Architecture

We describe the architecture of the planning system in [59].

We are currently implementing an architecture for CBP which consists of the modules described below.

Case Data Base: The Case Data Base (CDB) is a library of successful and unsuccessful situation/plan pairs (cases). These cases are stored in frame-like structures containing surface features (original raw data) and abstract features (inferences and generalizations) encoded in CRL. These features describe the situation (events, resource constraints, and goals) for which a plan was constructed. The library also maintains a record of our previous successful and unsuccessful attempts at modifying (sub-)plans.

Case Acquisition/Classifier: The Case Acquisition/Classifier Module is the keeper of the case library (CDB). There are many possible ways to generate the case structures. In our previous work with CARS [30], we acquired and stored cases using an existing GE conceptual information storage and retrieval system called SCISOR[93, 78] to tap on-line data bases containing unrestricted natural language descriptions of stories. This information was then stored as surface features of the case and interpreted by a rule-base which generated abstract features of the case. In other applications in which the information has already been organized, these frame-like structures can be derived from the schemas of the data bases used to store the raw input.

In either case, we determine taxonomic criteria for the representation, storage and classification of cases in the CDB. As part of this task we must determine the cases’ relevant and salient features, their values’ granularity, and their data structure and knowledge representation.

Case Indexer/Matcher/Retriever: This module takes a (possibly partial) description of a situation (referred to as the probe), and returns the most similar case(s) from the CDB, according to a measure of similarity based on relevant and salient features. This measure of similarity is used to rank and select the closest case(s) to the probe. The most salient features of a case for initial retrieval in case based planning, are its top-level goal(s), and its available resources.

Similarity Measure: Encoding cases in CRL allows similarity to be measured based on levels of abstraction between salient features in the domain knowledge hierarchies. We have designed multiple metrics for abstraction based similarity, and we plan to run experiments comparing their usefulness for different retrieval tasks (after collecting approximately 30 cases in the transportation domain). Currently we are using the most strict metric, which prefers abstraction up the hierarchy (i.e. a class is more similar to its parent and grandparent than to its sibling).

Case Analyzer: Once the retrieved cases have been compared with the probe, the “best” case(s) are sent to the Case Analyzer Module. This module generates a difference analysis between the probe and the retrieved case. This analysis consists of the most relevant
similarities among abstract features, used to justify the retrieval of the case, the most relevant differences among abstract features, used to identify missing pre-conditions that could disable parts of the associated retrieved solution (plan), and the goal similarity and differences, used to guide the adaptation and repair rules.

The choice of case(s) to be passed on to the case modifier is based on the needs of the reasoner, and the context of the probe. An evaluation of the similarities and differences and how they affect the ability of our system to modify plans is used to determine the case to be used by the reasoner. This information is displayed to the user by the Dialog Manager for verification, possible interactions and user-guided selections.

Case Modifier: The Case Modifier identifies all the parts of the retrieved solution (plan) which are not applicable or repeatable, because of a lack of "resources" noted in the difference list generated by the case analyzer. The case modifier proceeds to individually adapt these parts by using substitutions (e.g., replacing a sub-plan for a sibling node in a goal-plan taxonomy), compressions (e.g. eliminating the step from the plan and substituting dependent sub-plans), extensions (e.g., generalizing another sub-plan to cover and replace the current one), and other possible strategies.

Another possible way to adapt a plan is to recursively use case-based reasoning on the sub-plan. By indexing on the inapplicable sub-plan and its associated sub-goal, we can screen the other retrieved cases (or if necessary, the case data-base) to see if the same or similar sub-goal has been achieved by other sub-plans or if this specific sub-plan has been successfully modified in the past.

Case Projector/Evaluator: The modified plan is passed to a Case Projector/Evaluator, which tries to predict the success or failure of the modified plan by projecting it in time/space and by evaluating its relevant performance functions. This projection/evaluation can be done by a simulating the plan execution or by performing a theoretical analysis of its characteristics (e.g. throughput analysis of a network).

Beside determining if all the constraints have been met, this module also produces a cost of the entire plan (degree of success or failure) identifies possible sources of failures, and generates a prioritized list of sets of resource-goal constraints among which tradeoffs must be performed. For the military transportation planning domain, the case projector/evaluator will be implemented by scheduling algorithms and other analytical techniques.

Case Repairer: The output of the case projector, augmented by the difference analysis generated by the case analyzer, is the input to the Case Repairer. This sub-module uses a specialized knowledge base to determine if:

- it can accept some of the tradeoffs of the resource/goals constraints (thus considering the plan to be successful and propagating the modified constraints to the other sub-plans), or
- the plan needs to be returned to the case modifier to attempt a different adaptation rule, or
- the originally retrieved plan cannot be successfully modified and another similar plan must be selected for adaptation.

These resources include such things as differences in geographic features, lack of physical resources like planes, etc.
4.3 Goal Resolution

The resolution of the goals after the plan extraction stage is done using the information in the stored cases. The plan for an unresolved goal $g_i$ can be determined in one of the three ways: by identifying a suitable event for the goal $g_i$, by identifying an interpretation for goal $g_i$, or by resolving the subgoals of $g_i$. If a suitable event is found for the goal $g_i$, then a planned action is made for this goal using that event. This planned action is added to the strategy which had $g_i$ as its unresolved goal. If no suitable event is found then the next step is to identify a plan step interpretation for this goal $g_i$. If a plan step interpretation is found for this goal $g_i$, then a strategy is made using the interpretation. This strategy is added to the plan of the strategy which had $g_i$ as its unresolved goal. The goal of this strategy is the goal $g_i$ and its planned actions are made using the events that are part of the identified interpretation. In case no interpretation is found, then the subgoals of the goal $g_i$ are resolved. Before starting to resolve the subgoals, a new strategy is made whose goal is goal $g_i$. The plan of this new strategy will be the plan found by resolving the subgoals of goal $g_i$. If the goal node of goal $g_i$ is of type AND then all its subgoals have to be resolved. If the goal node of goal $g_i$ is of type OR then one of its subgoals has to be resolved. This process is repeated until we resolve the goals or the unresolved goals have no subgoals. If any goal is still unresolved then that goal is added to the list of goals with no possible plans.

(resolve-goal SWEETEN-DEAL)
WARNING: No event for the SWEETEN-DEAL goal
WARNING: No interpretation for the SWEETEN-DEAL goal
The goal SWEETEN-DEAL is an AND node
Find plans for achieving the subgoals (TR-SWEETEN-DEAL TA-SWEETEN-DEAL)
WARNING: No event for the TR-SWEETEN-DEAL goal
WARNING: No interpretation found for TR-SWEETEN-DEAL goal
Attempting to find plan for goal TR-SWEETEN-DEAL by planning for its subgoals
The sub goals of TR-SWEETEN-DEAL are: NIL
The TR-SWEETEN-DEAL goal is of type: OR
NO sub goals so it can not find plans for goal: TR-SWEETEN-DEAL
(NIL (#<PLANNED-ACTION SEEK-BUYERS-1140 5759CAE>))

Figure 4.3: The Resolution of SWEETEN-DEAL goal

The technique for resolving a goal $g_i$ can best be illustrated by following the steps taken during the resolution of a specific goal. The steps followed in resolving the goal SWEETEN-DEAL are given in Figure 4.3. No suitable event or interpretation from which a plan could be made is found for this goal. The subgoals TR-SWEETEN-DEAL and TA-SWEETEN-DEAL are then considered. For the subgoal TR-SWEETEN-DEAL, again no suitable event or interpretation is
found. The TR-SWEETEN-DEAL goal has no subgoal, therefore this goal can not be resolved and is added to the list of goals with no possible plans. For the subgoal TA-SWEETEN-DEAL, an event is found in the RJR-Nabisco case. Using this event a planned action #<PLANNED-ACTION SEEK-BUYERS-1140 5759CAE> is made and is added to the plan for goal SWEETEN-DEAL.

In the following sections we describe how the suitable events and interpretations are identified.

4.3.1 Identifying an event for a goal

The identification of an event for a goal $g_i$ is done by making a set $P_i$ of possible events, making an information tuple for each event in $P_i$, and identifying an event based on the lower bound of probabilities of the actions of events in $P_i$.

First, the set $P_i$ of possible events is made using the goal $g_i$ as an index. Instantiation links for goal $g_i$ are used to retrieve all the past instances of the goal. Only those goals are kept whose plan type is : event. Using the plan-goal links of the selected goals all the events that have been executed in the past cases for executing this goal are retrieved. The events in the list events-to-ignore are removed from retrieved events and the rest of the retrieved events form the set $P_i$ of possible events.

Then, the information tuple for each event in $P_i$ is made. This information tuple contains the action, the belief that the action achieved the goal, and the context of the action. The belief for this action that it did achieve the goal $g_i$ is the minimum of the belief in the goal and the belief in the goal achievement. The belief in the goal is obtained from the plan-goal link which contains the belief that the actual goal is the indicated goal. The belief in goal achievement is obtain from the belief in the causal link between the event and the changed state value. The changed state value which is considered for this belief is of the value of the state variable indicated by the state-var slot of goal $g_i$. As described earlier, state-var for the goal contains the state variables that will change if the goal is achieved. The context of the action is the context of the event for which the information tuple is made.

Finally, the identification of an event is made by first selecting an action and then selecting an event with that action. The action with the highest lower bound of probability is selected from all the actions of the events in $P_i$. The lower bounds of probabilities of the actions is determined using the information tuples made for the events in $P_i$. After selecting the action, the events in $P_i$ which have instances of the selected action are grouped. Among the grouped events is the event with the highest situational similarity that is identified as a suitable event for the goal $g_i$.

This approach for identifying the suitable event does not rely on domain knowledge. It uses the goal hierarchy for indexing into possible events and then uses the contextual information of the events and their similarities to the current situation to select the event. This approach shows how an event for making a plan step can be selected by leveraging the information in the past cases and how we can complement our weak domain theory by these cases.
4.3.2 Identifying an interpretation for a goal

The identification of an interpretation for a goal \( g_i \) is done by making a set \( P_{in} \) of possible interpretations and identifying an interpretation based on the situational similarity.

First, the set \( P_{in} \) of possible interpretations is made using the goal \( g_i \) as an index. Instantiation links of goal \( g_i \) are used to retrieve all the past instances of the goal. Only those goals are kept whose plan type is :interpretation. Using the plan-goal links of the selected goals, all the interpretations that have been executed in past cases for achieving this goal are retrieved. Among the retrieved interpretations, the interpretations of type plan step are selected. The interpretations in the list interpretations-to-ignore are removed from the selected interpretations and the rest of the selected interpretations form the set \( P_{in} \) of possible interpretations.

Then, the identification of interpretation is made. The interpretations in the set \( P_{in} \) are ranked by their situational similarities. Among the group of interpretations that have the highest similarity, the interpretation which has the highest level-of-typicality is selected. This level-of-typicality is used by experts to indicate how good this subplan is for the indicated goal.
Chapter 5

Evaluation and Results

5.1 Methodology for Evaluating the Generated Plans

The method used for evaluation of the generated plan within the M&A domain [2]:

The evaluation of the plans generated by CARS was done by comparing the plans generated by it for a given initial state of the case with the actual plans in that case. To evaluate the plan generated for a stored case \( C_s \), three steps were followed. First, a plan was generated for the goals of an agent in this case \( C_s \). Then, this generated plan was represented as a planned case \( C_p \) using CRL. Finally, the similarity between the events of the planned case \( C_p \) and the stored \( C_s \) was determined.

5.1.1 Generate plans for an agent

The stored case \( C_s \) was removed from the case library. A probe case was made for this case. The probe case only contained the initial state of the stored case \( C_s \). The most similar case was retrieved for this probe case. This retrieval was done on the ranking of the cases based on the situational similarity because the probe case only had the initial state. The plans for the goals of one of the agents in case \( C_s \) were then generated.

5.1.2 Represent the generated plan as a case

The generated plan was represented as a planned case \( C_p \) using CRL. The initial state of this case was the same as the initial state of the stored case \( C_s \). The planned actions of the generated plan were represented as events. The goals of the planned actions were represented as the goals of the events with degree of belief as *certain*. The events for the core planned actions in a strategy were linked together by the causal links with degree of belief as *certain*. A unique state was used as the context of the events except for the events of the planned actions whose with-action slots were not empty. The events of these planned actions had the same context state. For example, if planned action \textit{buy} had planned action \textit{sell} in its with-action slot, then the two events \textit{buy} and \textit{sell} for these planned actions had the same state as their context. The events \( E_{a_i} \) and \( E_{a_i} \) for the planned actions \( P_{a_k} \) and \( P_{a_i} \), where \( P_{a_i} \) appeared in the after-action slot of the planned action \( P_{a_k} \), were linked via causal and enable links. A new state \( S_n \) was defined. A causal relation from the event \( E_{a_i} \),
to new state $S_n$ was defined with degree of belief as "certain." This state $S_n$ was then added as an enabling state of the event $E_n$. The plan-steps interpretations were made for all the strategies in the generated plans. The goals of the strategies were defined as the goals of the interpretations. Initially, the interpretations were made for the strategies which had planned actions. For each strategy $S_i$ which had planned actions, a plan-steps interpretations $I_i$ was made. The events for the planned actions in strategy $S_i$ were then made as the parts of the new interpretation $I_i$. The strategies that were part of another strategy were then grouped into another plan-step interpretation. The top most interpretations were then grouped in the meta-strategy interpretation. The goal of this interpretation was the top level goal of the agent in the probe case.

5.1.3 Compare the events in the planned case and stored case

The events in the stored case $C_s$ were compared with the events in the planned case $C_p$ in a manner similar to the dynamic similarity, but with some differences. The main differences were that the similarity was done based on the actions rather than goals and the grouping of events in both cases were done using interpretations. We will next discussed how events were grouped into sub-plans followed by the discussion of how similarity was determined between these sub-plans.

Group events into sub-plans

All the events in the stored case $C_s$ are retrieved. Only the events whose agent has the same role as the agent for which the plan was generated are selected. The enabling events, compensatory events and reactory events are removed from these selected events. The selected events in the case $C_s$ and all the events in the case $C_p$ are divided into their phases (i.e., phase-2 and phase-3) because the comparison of events in a sub-plan is done within each phase.

The plan-step interpretation indicates the events that belong to an identifiable plan. From the set of events in a phase, using the membership links, all the plan step interpretations of these events are retrieved. This gives the sub-plans in a phase in both the cases. The grouping of events for these sub-plans is then done in the following three steps. First, for each interpretation, a set of events is made. An event may be a member of more than one set, when it is a part of more than one interpretation. Then, the events that are causally related within each set are grouped together. This grouping identifies the core events in a plan. Finally, the events in each group are further grouped together based on the states in which they were executed. All events executed at the same state are grouped together.

Similarity between sub-plans

The similarity between the plans in a phase were determined hierarchically based on the groupings of the events. Firstly, the similarity between the events grouped by state was determined. Secondly, the similarity between the sub-plans was determined. The similarity

---

1The reader will recall that in dynamic similarity we used interpretations for grouping the events in the retrieved cases and relationships for grouping of the events in the probe case.
between the sub-plans was based on the aggregation of similarities between the groups of causally related events in that sub-plan. Finally, the similarity between the sub-plans were aggregated to get a phase level similarity.

**Similarity between events grouped by state:** The similarity between the events is based on the actions of the events unlike the similarity between events grouped by state in dynamic similarity determination, where the similarity is based on goals. The reason for this is that the plans in the two cases here are for the same goals at the sub-plan level and what we want to determine is how similar these plans are. To determine the similarity between two groups of events, where each group has events at the same state, only the similarity between the actions of events in these two groups was considered. The similarity between these two groups of events was determined by computing the similarity between two lists of actions where each list of actions represents the actions of events in a group.

The similarity between the lists was determined by computing a _pair wise similarity of actions_ in the two lists and then _aggregating_ them. This pair wise similarity and aggregation was done for all the combinations of actions and the best result was considered as the similarity between the two lists.

The _pair wise similarity_ of two actions was obtained by doing _similarity by abstraction_ on the action hierarchy. This similarity by abstraction returns a similarity value, which is based on how close two actions are in the action hierarchy. For same actions the similarity value returned is *complete-match*, and for siblings it is *almost-complete-match*. The certainty of this similarity value is based on the certainties of the actions. In our case these certainties of the actions are *certain*. For example, the similarity between the actions MAIL-CONTACT and TELEPHONE-CONTACT is based on their relation in the action hierarchy. Both these actions are direct children of CONTACT-ACTIONS. Therefore, their pair wise similarity is *almost-complete-match* with certainty *certain*.

If the two lists had different number of events then the difference was considered as *no-match*. The certainty of this similarity was (0 1.0) representing ignorance because we had no evidence as to what that missing event would have been. For example, if one list had five and the other had seven events then two *no-match* similarities with (0 1.0) certainty were added to the pairwise similarity list before aggregation.

**Similarity between sub-plans:** Pair wise similarities between the groups of causally related events in the set representing a sub-plan were computed and then aggregated. This process was performed for all the combinations of the causally related groups in the two sets. The similarity between the two sub-plans was the best similarity of all these combinations.

To determine the similarity between the groups of causally related events, again pair wise similarity between events was computed and then aggregated. It was only done for the combinations which maintain the order, and then the best result was taken. Here, while determining the combination, the _order of execution was maintained_. For example, consider two groups of causally related event, \((e_1, e_2, e_3)\) and \((e_4, e_5)\). A combination maintaining the order will be \(((e_1, e_4), (e_3, e_5))\). A combination that _does not_ maintain the order is \(((e_2, e_4), (e_1, e_5))\).
Similarity between events at the same phase: The similarity at the phase level was based on the similarity between the sub-plans. The pair wise similarity between sub-plans in the phase was computed, as discussed in the last section. If the top level goal node of the agent was of type AND then *no-match* with certainty (0 1.0) was added to the pair wise similarity list for each direct child goal with a missing sub-plan. The pair wise similarity was then aggregated. Again this process was performed for all combinations of sub-plans and the best similarity was taken as the resultant similarity between the sub-plans in the two cases. The similarities between sub-plans and the similarities between the events at the phase level were used to determine how close the generated plans were to the actual plans.

5.2 Interpretation

We explain what is meant by interpretation of a plan in [59].

Interpretations are explanations for the occurrence of sets of events. \( M(\mu) \) is a link with associated certainty level which includes an event in an interpretation. The certainty level indicates the degree of belief in the inclusion. Interpretations may contain some local knowledge such as a goal, objects affected by the interpretation, etc.

Example:

```
TRANSFER-CARGO-001 is a TRANSFER-CARGO
    from: WAREHOUSE-001
    to: WESTOVER
    cargo: (contents WAREHOUSE-001)
    goal: `(((location cargo) to))`
    parts: `(LOAD-001 TAKEOFF-001 LAND-234 UNLOAD-23)`
```

Some example interpretations are constraint maintenance, cargo transfer, etc. The interpretations are currently used to represent the case analysis of a domain expert. At some levels these interpretations can be seen as steps in the hierarchical expansion of a plan and can be used to augment the goal/plan hierarchy.
Chapter 6
Conclusions and Summary

This report has described work done in the area of case based planning in several domains. Ayub [2] summarizes for us the contribution made in the field of M&A.

The goal of this thesis is to apply AI planning techniques to complex and real world situations. In these situations, the expansion in the problem occurs along various dimensions, such as the lack of a complete domain theory, the inherent uncertainty present in the information used in planning, and the dynamic nature of the planning process. In this thesis we address each of these dimensions.

First, we investigate how to develop a strategic plan when we do not have a good model of the world in which actions are going to be applied. Lacking a good domain theory to model the real world, we resort to using past cases to guide us in the development of strategic plans. Then, we study the development of plans using information from the past cases which may have uncertainty. Uncertainty in planning can be of two types: structural and parametric. Structural uncertainty occurs in various mappings and parametric uncertainty occurs in various assignments of the state variables. Finally, we investigate the representation of the dynamic nature of the planning process. This planning process is observed as the changing world states and the actions executed by the agents.

Our proposed planning system, named CARS (Combined Approximate Reasoning System), is rooted on Case-Based Planning (CBP) techniques. The main contribution of this research is the development of a CBP approach that uses cases to supplement its weak domain theory. This is the first case-based reasoning system that reasons with cases that are dynamic and have uncertain information in their case features.

We tested our system in the domain of Mergers and Acquisitions (M&A). The techniques we have developed for reasoning can be applied to planning problems in other domains. Our reasoning techniques rely on the conceptual and episodic knowledge of the domain. For another domain, these two types of knowledge can be defined using our representational language (CRL). The planning can then be done in this new domain using our techniques.

The specific contributions of this research to the subfields of case-based planning are summarized in the rest of this section. For each subfield, first we identify our contribution, then we summarize how it is achieved, and finally we justify why it is our contribution.

- **Case representation**: Development of a Case Representation Language (CRL) to represent cases that evolve over time (dynamic cases) and have both structural and parametric uncertainty.
CRL uses a network of states and events to model the case evolution. The sequence of states represent the changing states of the world in the case. The sequence of events represents the encapsulation of state changing actions with their contextual information encoded as links such as: membership, causal, enable. The structural uncertainty is modeled as the degree of belief in the links representing the mappings. The parametric uncertainty is modeled as the membership of the label, representing an ill-defined value, in a fuzzy set.

Our work on case representation is a contribution to the field of case-based planning as no other reasoning/planning system represents cases that are dynamic and have uncertain information. The cases in our system are the actual episodes of what happened unlike other CBP systems whose cases are specific plans for each situation.

• Similarity assessment: Our contributions to the field of similarity assessment are:
  
  – The determination of similarity between dynamic cases based on the aggregation of the situational and dynamic similarities.
  
  – The determination of pairwise similarity as a function of the fuzzy distance between two objects in the pair.
  
  – The determination of dynamic similarity based on the goals of actions causing the case evolution.
  
  – A flexible approach to aggregating similarities of partial matches.
  
  – The Propagation of the uncertainty in case information to the uncertainty in similarity.

The process of determination of situational similarity uses the state variables that define the situation of the case. These state variables are used to get abstract (derived) features. The similarity of each abstract feature is computed as the complement of the distance between the fuzzy numbers representing the feature values. The abstract features similarities are aggregated hierarchically, according to a semantic taxonomy, to get situational similarity. The aggregation is based on T-norms, averaging operators, and T-conorms. The parametric uncertainty is propagated through all steps of the similarity computation to get the certainty in the determined value of situational similarity. The process of determination of dynamic similarity uses the state changing actions that cause the evolution of the case. The goal links are used to get the goals of these actions. Goal estimation is done for the actions that are not known. The actions are grouped using their contextual information encoded as links such as: membership, causal, enable and context. The similarity of each group is computed based on the relationship between goals of the actions in the group. The grouped actions similarities are aggregated hierarchically to get dynamic similarity. The structural uncertainty is propagated through all steps of the similarity computation to get the certainty in the determined value of dynamic similarity. The emphasis on certain values during similarity computation and/or leniency/strictness in similarity aggregation is represented by varying the range of operators, the termsets, and the weights.
No other case-based reasoning system uses dynamic similarity in case similarity assessment. Therefore our approach to determining dynamic similarity is an original contribution to this problem. No other system considers the uncertainty in the case features values and thus can not reflect it in their determined similarity. Our system is the first system that uses a flexible approach to aggregating similarities that also takes into account the uncertainty in the similarities.

• **Goal estimation:** Development of an approach to estimate the goal of an action using stored cases.

  The process of goal estimation of an action uses the action’s past goals to define the possible goal space for determining the goal and belief in the determined goal. The probability distribution of the goals in the defined goal space is determined based on the lower bound of the probability of each goal. The lower bound of the probability of a goal is the aggregation of the contributed mass of that goal for each of the past executions of the action. The contributed mass of a goal, when it is among the indicated goals of an executed action, is determined as the product of the belief that the goal is the actual goal of the executed action (structural uncertainty) and the relative mass of this executed action’s context. The situational similarities between the context of the past executions of the action and the current context are normalized, and the relative mass of an action’s context is that normalized situational similarity. The degree of belief in the determined goal is computed as a function of the entropy, relative entropy, and the cardinality of the determined probability distribution.

  Our system is the first case-based planning system that tries to estimate the goals of input actions using cases in its case memory. All other CBP systems assume the goals to be known.

• **Plan adaptation/modification:** Our contributions to the field of Plan adaptation are:

  - The identification and extraction of a plan from past cases.
  - The development of an approach for determining an appropriate plan for a goal using previous cases.
  - The development of an approach for Case-based adaptation which only uses the semantics of the features of CRL for modifying plans.
  - The use of multiple paradigms (CBR and RBR) in plan identification and adaptation.

  The process of plan adaptation/modification uses the cases in the case library that define the episodic knowledge to modify the plan extracted from the retrieved case. The extraction of a plan also involves the identification of resources and constraints which are achieved by combined reasoning (CBR and RBR) on the retrieved case and the current situation. The planners’ goals which do not have plans in the extracted plan are resolved. The resolution of each goal is done by identifying for it a suitable plan in one of the cases in the case library. The identification of the plan for a goal is based on
the situational similarity of the context for that plan’s actions with current context and
the structural uncertainty in the mapping of the plan to the goal. The plan extracted
from the retrieved case is augmented with the identified plans for the unresolved goals.
This plan is then structurally adapted to the current situation. Structural adaptation
is done on the slots of the planned actions using the adaptation information attached
to the action classes in the action hierarchy. Case-based adaptation is done on the
planned actions that can not be executed in the current context. This adaptation
attempts to find a suitable action whose execution will change the state of world such
that the planned action can be executed in it. This is again based on the situational
similarity of the context of past actions and the structural uncertainty. In case no
suitable action is found, then an alternate plan is developed for the subgoals of the
goal of the planned action.

We do not assume that each case represents a plan, so our approach to extracting
and identifying a suitable plan adds to the capability of any existing CBP system. We
determine appropriate actions for a goal by considering all the past actions for this goal,
their contextual similarities to the current context, and their structural uncertainty.
Our approach is unique in the sense that we leverage on all the past experiences for
achieving a certain goal when we are planning for that goal. All other CBP systems rely
on domain knowledge for adaptation while we do most of the adaptation using cases.
The Case-Based adaptation techniques of other systems rely on domain knowledge
encoded in some form, while we use the features of the CRL to make appropriate
adaptations. We use the structural uncertainty in cases for adapting our plan while
other systems do not consider structural uncertainty at all.

- **Integrated reasoning methodologies:** The integration of a rule based reasoning
(RBR) system with a Case-based reasoning (CBR) system where the dominant reasoner
is CBR.

The case-based reasoning cycle uses rules that define general domain knowledge at
various stages of reasoning. The rule-based reasoning with these rules is done using
PRIMO. The parts of the problem whose solution can to be determined using RBR are
declared in the RBR interface and PRIMO rules are developed for them. When the
system encounters a subproblem which is declared in the RBR interface, then it invokes
PRIMO for the solution. This solution is then used by the Case-based reasoner.

Our system is the only system that uses the Case-Based Reasoning technique as
the dominant reasoning technique and has a RBR system integrated with it so as to
leverage the general domain knowledge whenever it is available.

We summarize the CRL in [59].

In this paper we have described the design and implementation of the Case Representation
Language (CRL), a language designed to represent dynamic cases. We have illustrated CRL
with examples from the military transportation domain, which will be the focus of our CBP
efforts in the forthcoming future. We have described the design of our CBP architecture and
its partial implementation.

Our preliminary results in using CRL and a subset of the architecture modules have
confirmed the soundness and the representational adequacy of CRL for case storage, case
retrieval, case analysis, and solution adaptation. Our future work will be focused on completing the implementation of the CBP architecture, expanding our case library with a variety of military transportation cases, and continuing the testing and validation of our CRL implementation.
Part II

Temporal Reasoning
Abstract

We describe a constraint-based model for representing and reasoning about qualitative and quantitative aspects of time. Our model allows substantial expressiveness, provides fast computation over convex intervals, and will serve as a testbed for heuristic topology-driven techniques for handling calculations over non-convex intervals. We describe an implementation of this model that features a graphical interface using X-Windows and InterViews. We anticipate that this model and its implementation will find applicability in several areas, including scheduling, project planning, feasibility analysis, and spatial/temporal databases.
Chapter 7

Introduction

We will describe the development of ontology, algorithms, and software to provide effective, efficient temporal reasoning capabilities critical to applications such as scheduling, project planning, feasibility analysis, and spatial/temporal databases. In particular, we will describe Tachyon, a prototype software tool for constraint-based temporal reasoning. (A tachyon is a theoretical subatomic particle capable of traveling faster than the speed of light, perhaps even traveling backwards through time. The name is chosen for its relation to time, and our emphasis on performance.) One key reason for developing Tachyon is to serve our need for a test-bed for evaluating new methods of coping with disjoint constraints as they appear in the transportation planning and scheduling domains. Tachyon also has the potential to be used as a more sophisticated project planning tool than the likes of MacProject, a popular tool available for the MacIntosh. It also promises to be useful in scheduling problems such as satellite and telescope use. We have integrated an early version of this work into GE-CRD's plan recognition program Patti++, where it was used to validate temporal sequencing of events as an aid in formulating plan hypotheses. We have also integrated the Tachyon event model and query capabilities into a geographic information system (GIS) package under development at GE Aerospace.

7.1 Motivation and Applications

Reasoning about time pervades our daily lives. Few of us are free of schedules, appointments, and deadlines. Temporal reasoning needs are also critical in many computer applications: databases, simulators, expert systems, and industrial scheduling and planning systems. Each of the following examples need to manipulate temporal information to model the world.

The productivity of assembly lines is inversely proportional to the total down time, that is, all intervals during which a station or tool is idle or being prepared for the next job. Idle time can be reduced by such methods as altering the route a job takes through a line or by maximizing the use of limited resources. When a tool requires maintenance between jobs, e.g. changing bits on a drill press, and the maintenance varies depending on the nature of the job, the total maintenance time becomes highly-dependent on the order in which the jobs are scheduled. By minimizing down time, we can increase the productivity of an assembly line.
Figure 7.1: A tool using spatial/temporal queries to access a database

Plan deployment often involves complex interaction between critical steps in the plan. For instance, in a relief exercise, people must be available to unload a cargo plane. If they arrive too early, their time is wasted; if they arrive too late, the pilot’s time is wasted. Either way, the efficiency of the operation is reduced, and the separate paths of the plane’s schedule and the relief-workers’ itinerary are affected. Separate branches of an overall plan should interact smoothly. All such interactions should be carefully considered and specified accurately.

The GIS system in Figure 7.1 keeps spatial and temporal information on a number of objects in the world ranging from locations of mobile vehicles at different times, to permanent positions of cities, buildings, and mountains. In this example we have selected a region of interest, and request retrieval from the database of all vehicles that could possibly have been in that region between June 7th and June 23rd, 1992. As such databases can be extensive, performance in retrieval is critical.

Natural language processing requires understanding of temporal and tense information in order to answer queries about the sentences. Consider a search for discussion of Japanese economics during the mid-seventies. The query should find historic references in articles published after 1980, as well as “current” reports during the 1970’s. Another query re-
quiring temporal reasoning would be a request for the names of all senators who served as representatives for at least two terms before being elected to the senate, where they died in office during their first term.

Problems such as those cited above become difficult due to the number of possible configurations the solutions may take. By developing an automated means of expressing the system and its constraints, high-speed computers using efficient algorithms can outperform humans in their ability to both digest a voluminous amount of data and produce optimal solutions to the problems.

To communicate the system and its constraints to a computer, a sufficiently expressive language/representation must be constructed. The model should easily map to the real-world problem so the user can perceive the interactions, both while specifying the system and interpreting the solution. The effects of decisions made by the computer (e.g., ordering) should be evident in the numbers returned by the system when all constraints have been considered and propagated.

Time is a precious, nonrenewable resource. Operating costs directly reduce profits. We can realize substantial efficiency gain by finding optimal solutions to problems where these resources are critical. Computers greatly outperform humans in organization and number-crunching skills, so exploiting this ability in the field of temporal representation and reasoning shows great promise.

7.2 Background

Temporal reasoning and representation issues have provided fertile ground for research for many years, and many unresolved questions remain. There exists a wealth of literature on the topic, aspects of which have been studied by philosophers, linguists, computer scientists, operations researchers, and artificial intelligence researchers. Research that is particularly relevant to the work presented in this document is the interval algebra of Allen [10, 11], the philosophies of Van Benthem [25], and the models and methods of Dean & McDermott [15], Dechter, Meiri, & Pearl [16], Rit [22], Valdés-Pérez [23], and Vilain [26].

In the following, we will call the basic objects in temporal reasoning “events”. Events are related to one another by symbolic and numeric constraints. A temporal reasoning system evaluates the events in the context of the constraints and derives solutions that conform to all specified requirements.

An event might be a fact, execution of a task, or a simple time stamp, depending on the level of abstraction. We will consider any proposition with temporal extent an event in this discussion. Occurrences of events are one-dimensional segments in time. Repetitive events which occur more than once (e.g. having lunch each day) are not explicitly part of the paradigm we will be exploring; we can model them by decomposing them into individual occurrences (e.g. Lunch on Aug 4th, 1992). As events are often shown as nodes in a graph, we may use the term node and event interchangeably.

We usually derive relationships between events from more abstract descriptions of the world, such as a project plan. These descriptions may indicate an ordering (e.g. A is before B), or show a relationship between two intervals (e.g. A is before or during B), or specify a quantitative separation between the events (e.g. A is between 2 to 4 hours before B).
Qualitative relationships allow simple orderings of events, while quantitative relationships can enable the model to explicitly find occurrence times for the events. Relationships or “constraints” correspond to edges when a graph representation is used, thus we may use the terms relation, constraint, and edge interchangeably.

The most basic way to represent time is to label all instants with an absolute time stamp. Each such event corresponds to an instant in time. An alternative to this is to provide a full, linear ordering, without any mapping to a clock or scale. Instantaneous events whose placement in time is uncertain can be constrained within a range of values (e.g. 9am to 10am) corresponding to the earliest and latest possible time of occurrence. Allen based his well-known temporal relation calculus and propagation algorithm [10, 11] on this model of using intervals to represent the time over which an event may occur. This representation allows expression of $2^{13}$ possible constraints between two intervals. Though an incomplete, local constraint propagation can be accomplished in $O(n^3)$ time, any complete algorithm using this expressiveness and guaranteeing consistency verification (or constraint satisfaction) is NP-hard [27]. Such intractability extends even to simple (qualitative, disjoint) models [13]. VanBeek & Cohen [24] went on to enumerate a subset of these constraints which allow polynomial solution of such systems. There are techniques, such as path consistency which can simplify networks by imposing local consistency as a preprocessing step to constraint satisfaction. Path consistency has been the favored approach, explored by Allen [11], and others [16, 24].

The model is further complicated when we recognize that real-world events with durations map more directly into a single event than a start event, plus a finish event. Dean & McDermott [15] developed a representation using exclusively duration constraints. All information in their model is duration between time points. Gantt charts, used in scheduling and project management, provide a familiar example of such events. An acyclic directed graph provides a partial ordering over such events. The world-events each have a specified start and end event, constrained by the duration of such an event (e.g. Lunch takes 30 to 60 minutes). This model is useful in applications such as scheduling where event durations are known. In the same manner in which intervals are used to denote uncertain event times, uncertain length of the duration can also be specified as an interval.

Temporal constraint networks typically use quantitative values to express the allowable relationships between events. In our work we have also conformed to this, using a “point-based” representation of time. The other prominent representational paradigm for time is found in James Allen’s interval calculus. Allen [11] developed a set of qualitative linguistic values for describing relationships between events. Qualitative constraints allow one to specify relationships between events using linguistic descriptions, without numeric bounding. For instance, we may want to express abstract temporal ordering on a delivery route by saying that store A is visited before store B, without specifying when either delivery is made or numerically constraining the time between the two deliveries. Deviation from that sequence should be identified as causing temporal inconsistency. A qualitative network is inconsistent when there exists an unresolvable conflict between instantiated variables and their constraints. In the above example, specifying A before B, but giving B earlier times than A results in inconsistency. Table 7.1 shows a listing of qualitative temporal relations.

A constraint is convex if the allowed distances between the constrained event-variables form a continuous interval. Projected into dimensions for each related variable (in the six-
Table 7.1: Relations Expressible in Allen's Calculus

The relation $x \rightarrow y$ (e.g., before)–$y$ is illustrated by the relative positions of the intervals in this table. Simultaneous starts and finishes are indicated by vertical end-brackets $(\triangleright)$, while the angular end-brackets $(\triangleleft)$ indicate otherwise. Line length represents relative duration of the events.

<table>
<thead>
<tr>
<th>before</th>
<th>$\triangleleft x \rightarrow$</th>
<th>after</th>
<th>$\triangleleft x \rightarrow$</th>
</tr>
</thead>
<tbody>
<tr>
<td>meets</td>
<td>$\triangleleft x \rightarrow$</td>
<td>methby</td>
<td>$\triangleright x \rightarrow$</td>
</tr>
<tr>
<td></td>
<td>$\triangleright y \rightarrow$</td>
<td></td>
<td>$\triangleleft y \rightarrow$</td>
</tr>
<tr>
<td>overlaps</td>
<td>$\triangleleft x \rightarrow$</td>
<td>overlappedby</td>
<td>$\triangleleft x \rightarrow$</td>
</tr>
<tr>
<td></td>
<td>$\triangleright y \rightarrow$</td>
<td></td>
<td>$\triangleleft y \rightarrow$</td>
</tr>
<tr>
<td>starts</td>
<td>$\triangleright x \rightarrow$</td>
<td>startedby</td>
<td>$\triangleright x \rightarrow$</td>
</tr>
<tr>
<td></td>
<td>$\triangleright y \rightarrow$</td>
<td></td>
<td>$\triangleright y \rightarrow$</td>
</tr>
<tr>
<td>during</td>
<td>$\triangleleft x \rightarrow$</td>
<td>contains</td>
<td>$\triangleleft x \rightarrow$</td>
</tr>
<tr>
<td></td>
<td>$\triangleright y \rightarrow$</td>
<td></td>
<td>$\triangleleft y \rightarrow$</td>
</tr>
<tr>
<td>finishes</td>
<td>$\triangleleft x \rightarrow$</td>
<td>finishedby</td>
<td>$\triangleleft x \rightarrow$</td>
</tr>
<tr>
<td></td>
<td>$\triangleright y \rightarrow$</td>
<td></td>
<td>$\triangleleft y \rightarrow$</td>
</tr>
<tr>
<td>equals</td>
<td>$\triangleright x \rightarrow$</td>
<td></td>
<td>$\triangleleft y \rightarrow$</td>
</tr>
</tbody>
</table>

Table 7.1: Relations Expressible in Allen's Calculus

The relation $x \rightarrow y$ (e.g., before)–$y$ is illustrated by the relative positions of the intervals in this table. Simultaneous starts and finishes are indicated by vertical end-brackets $(\triangleright)$, while the angular end-brackets $(\triangleleft)$ indicate otherwise. Line length represents relative duration of the events.

tuple), the interval will enclose a convex polygon. Some problems require expression of non-convex, disjoint\(^1\) constraints.

For example, consider a single-lane train track between A and B. There are two trains; train 1 is scheduled to travel from A to B, while train 2 will go from B to A. Clearly, they cannot use the track simultaneously. The constraint we need to state is “Train 1 will use the track before or after train 2.” Such disjoint constraints, which require performance of multiple tasks on a single vehicle/tool/machine without precedence constraints between the competing tasks, arise frequently in planning and scheduling domains. Unfortunately, introducing them may greatly increase the complexity of processing the network [27]. This is due to an explosion in the number of potentially valid combinations of constraints to be considered during solution. Another key purpose of our temporal reasoning environment is to serve as a test-bed for evaluating new methods of coping with disjoint constraints.

What we will call a solution to a system is either a response that the network is inconsistent or what is called the minimal network [20]. Montanari defines a minimal network as the least element of the set of equivalent, optimal approximating networks which conform to the binary constraints between sets of possible value pairs of variables. All global constraints that can be transmitted through all the possible paths in the network are explicit, and are equivalent to a solution of the set of linear equations formed by these variables.

\(^1\)Strictly speaking, many convex relations are disjoint. For this discussion, assume we mean non-convex relations when we say disjoint.
Chapter 8
Design Issues

8.1 Defining the Problem

While developing Tachyon, specification of requirements and a survey of literature helped us formulate desiderata for a constraint-based temporal reasoning system for planning and scheduling tasks.

It should be able to deal with uncertainty regarding the exact occurrence time and duration of occurrence of events. We do not always have complete or certain information concerning the events, but may still wish to instantiate the known values.

Some constraints are sufficiently expressed by linguistic values, while others need numeric distances separating the events. Constraints should be able to express both quantitative and qualitative relations between events, e.g., \(X\) is before or meets \(Y\), and \(X\) ends between 15 days before \(Y\) starts. They should be capable of expressing parameterized qualitative constraints between events, e.g., \(X\) is before \(Y\) by at most 6 days, and allow specification of disjoint constraints, e.g., 2-4 or 8 hours before.

We should provide data structures and algorithms for efficient storage and retrieval of temporal data. Queries on current event instantiations should be supported, e.g., "What events could possibly take place from 10:00am to 11:00am?". System variables should support different granularities of time units, e.g., seconds or days. We should be able to check the system for constraint satisfaction and propagate values to the events which satisfy the constraints.

To demonstrate the system and allow integration into other efforts and user communities, precautions should be taken in preparation for projected uses. The overall system should promote ease-of-use via graphical input and display capabilities, run as a subprocess in other applications as well as stand-alone, utilize techniques that will remain effective even in very large application domains, and serve as a versatile testbed for exploring new techniques for coping with the intractability associated with disjoint constraints.

Some models, e.g., Allen [11], have given special attention to persistence. That is, default reasoning such as "The computer is on" should remain true (persist) so long as we receive no contrary information ("Joe shut off the computer at 10:00am"). Given these capabilities, a notion of "current situation" is formulated. This work is not intended to address such questions. From a position out-of-time, we look at cross-sectional models of an entire system, projected into the past and/or future, with no explicit sense of "now."
8.2 Difficulties

We faced three main difficulties in providing the capabilities outlined above. The first is that consistency verification of a system supporting interval algebra (disjoint constraints between interval-based events) is NP-hard [13, 27], so heuristics must be applied to solve large systems.

The second problem, related to the first, is that when disjoint constraints are allowed, a large number of unique, consistent solutions may be found. Deciding which solution(s) to present to the user, and/or how to present more than one must be addressed. Some solutions might be preferable to others, though we currently have no means of specifying preferences.

The third difficulty is tracing what caused a system to become inconsistent. The constraints are usually specified at a much higher level of abstraction than the level at which they are solved, thus mapping the point of inconsistency to the original problem is non-trivial, and outside the scope of the current effort. Vilain [26] examined tracing the origin of inconsistency, but his work was performed on a much simpler event model. Dean & McDermott [15] use truth maintenance to address this issue.
Chapter 9

Results/Implementation

9.1 Temporal Constraint Networks

Although not without limitations, we found that the graph-based temporal constraint network (TCN) paradigm provided a good starting point for our research. This paradigm has also been explored by others, including Raúl E. Valdés-Pérez [23] and Dechter, Meiri, and Pearl [16]. Temporal constraint networks are a specialization of general constraint networks, formulated by Montanari [20]. A constraint network is simply a graph in which nodes correspond to variables and edges constrain the values the associated variables can be assigned. The constraints express binary relations between two variables. Assigning unique domain values to the variables is an instantiation. An instantiation satisfies a constraint if the variable assignments do not violate the constraint. A graph instantiation is consistent if it satisfies all the constraints of the network.

The simple TCN shown on the left of Figure 9.1 has three node-events (Eat Lunch, Coffee Break, and Eat Dinner), about which only duration is known. The edge-constraints shown indicate: Eat Lunch occurs no more than 2 hours before Coffee Break, Coffee Break ends at least 2 hours before Eat Dinner, and Eat Lunch will precede Eat Dinner by no less than 4 hours, and no more than 6. Once a time is given for the start or completion of any of the events, the constraints will narrow the possible times at which the other events can consistently occur; this narrowing is called propagation or tightening of the network.

There are several advantages to using TCNs to represent temporal relationships. These include easy visualization through graphical representation and the ability in some cases to use linear programming techniques to evaluate consistency and propagate information throughout the network. The TCN shown on the right in figure 9.1 demonstrates one type of TCN, that described in Dechter et. al. They call this the “Temporal Constraint Satisfaction Problem” (TCSP). The temporal distance between two events is shown as an interval, noted near the edge. The TCSP uses an interval representation for event times and constraints. This representation imbues events with the ability to express duration by time-stamping the start and the end times. The interval on the constraints allows for some uncertainty in the temporal distance separating events (e.g., 4-6 hours between Eat Lunch and Eat Dinner).
9.2 The Tachyon Model

To handle both uncertainty (in event occurrence) and duration, Tachyon represents events using 6-tuples, as described by Rit [22]. This representation satisfies several of our key desires, facilitating the job of mapping real-world event data to a single event in the model.

The event template shown in Table 9.1 represents the parameters of a Tachyon event. In order to represent the same uncertain information using TCSP intervals, an event must "artificially" be divided into a start event and a finish event, with a constraint between the two indicating duration. There are also occasions where duration is known, but no start of finish time information is available, e.g., templates: refueling always takes 5-10 minutes.

The 6-tuple from Table 9.1 allows a single network node to map to an entire real-world event, accounting for both duration and uncertainty. The added event expressiveness de-

<table>
<thead>
<tr>
<th>Earliest Possible</th>
<th>Start Time</th>
<th>Latest Possible</th>
<th>Start Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Earliest Possible</td>
<td>Finish Time</td>
<td>Latest Possible</td>
<td>Finish Time</td>
</tr>
<tr>
<td>Minimum Possible</td>
<td>Duration</td>
<td>Maximum Possible</td>
<td>Duration</td>
</tr>
</tbody>
</table>

Table 9.1: Event 6-tuple
mands similar expansion of the constraint model. Rit's work [22] and the model we used in the *Patti++* software were limited to using qualitative constraints.

Although we feel that a point-based representation is to be preferred for most of the temporal reasoning tasks we face, it is often convenient to allow relationships to be expressed using qualitative values, which are converted to numeric equivalents internally. This also allows some expansion in expressiveness of qualitative relationships, e.g., parameterized qualitative constraints such as *at least 2 hours before*, with no performance penalty.

Tachyon networks require numerical distances between events, i.e., quantitative constraints are needed. Quantitative constraints place numerical bounds on the temporal relationship between two events. For example, we should be able to express the constraint that a job can't be started on a given machine until some interval is allowed for changeover from the previous job. This interval is known (at least within some bounds) and any deviation from it should be found to be inconsistent.

We express qualitative constraints by introducing the notions of *epsilon*, the smallest distance possible, and *infinity*, the largest. For example, the qualitative relation *before* is interpreted as "There is a non-zero, positive distance between Event 1 and Event 2." Thus, we can say the distance between Event 1 and Event 2 is at least *epsilon*, at most *infinity*. In Tachyon, we can also expand on Allen's linguistic relationships by adding parameters to some of the relations. For instance, instead of simply saying we pick up our tickets *before* our flight, we can say we pick them up at least 1 hour before our flight. Parameterization is an option for the Allen relations *before*, *overlaps*, *overlapped by*, and *after*. Each of these is given the ability to take on two parameters, representing the minimum and maximum distance to which they refer. We must exercise care in introducing such parameterized qualitative relationships, however, as they can introduce intractability. Several convex disjunctions, e.g., as enumerated by VanBeek and Cohen [24], lose convexity when parameters are added. To illustrate this problem, consider the convex relation *before or meets*, meaning one event occurs zero to *infinity* time units before the other. If the user specified a minimum value for before (e.g., at least 10 before) then the relation is no longer convex. Thus, we carefully enforce convexity when possible.

<table>
<thead>
<tr>
<th>Minimum time between</th>
<th>Start&lt;sub&gt;1&lt;/sub&gt; and Start&lt;sub&gt;2&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum time between</td>
<td>Start&lt;sub&gt;1&lt;/sub&gt; and Start&lt;sub&gt;2&lt;/sub&gt;</td>
</tr>
<tr>
<td>Minimum time between</td>
<td>Start&lt;sub&gt;1&lt;/sub&gt; and Finish&lt;sub&gt;2&lt;/sub&gt;</td>
</tr>
<tr>
<td>Maximum time between</td>
<td>Start&lt;sub&gt;1&lt;/sub&gt; and Finish&lt;sub&gt;2&lt;/sub&gt;</td>
</tr>
<tr>
<td>Minimum time between</td>
<td>Finish&lt;sub&gt;1&lt;/sub&gt; and Start&lt;sub&gt;2&lt;/sub&gt;</td>
</tr>
<tr>
<td>Maximum time between</td>
<td>Finish&lt;sub&gt;1&lt;/sub&gt; and Start&lt;sub&gt;2&lt;/sub&gt;</td>
</tr>
<tr>
<td>Minimum time between</td>
<td>Finish&lt;sub&gt;1&lt;/sub&gt; and Finish&lt;sub&gt;2&lt;/sub&gt;</td>
</tr>
<tr>
<td>Maximum time between</td>
<td>Finish&lt;sub&gt;1&lt;/sub&gt; and Finish&lt;sub&gt;2&lt;/sub&gt;</td>
</tr>
</tbody>
</table>

Table 9.2: Constraint 8-tuple between *Event<sub>1</sub>* and *Event<sub>2</sub>*

Edge constraints are expressed internally by an 8-tuple in Tachyon, the semantics of which are described in Table 9.2. The Allen relations described in Table 7.1 are translated into this 8-tuple representation for consistency (as mentioned before).
We seek to merge the highly-expressive model found in Rit's work [22] with (temporal) constraint network techniques to produce a model with the best aspects of both: complete expression of an event in a single node, and high-performance computation on the system. The sample network shown in Figure 9.1 is shown in Figure 9.2 with its corresponding event and constraint values according to the tuple models described earlier. Note that unknown values are shown as infinite intervals.

### 9.3 Propagation of Constraints

On graphs consisting solely of convex constraints, or "chosen" convex constraints on non-convex constraints, Tachyon uses a modification of the Bellman-Ford shortest-path algorithm to propagate information and tighten the bounds of variables in the graph. This differs from Dechter et. al., who use the Floyd-Warshall algorithm. Descriptions of these can be found in Cormen, Leiserson, and Rivest [14]. Both algorithms have $O(n^3)$ time complexity, where $n$ is the number of nodes. In the testing we have done, the Bellman-Ford algorithm provided a substantial performance increase over the Floyd-Warshall algorithm, especially when the corresponding graphs are fairly sparse.

The Bellman-Ford algorithm solves the single-source shortest-paths problem where edge weights can be negative. The Floyd-Warshall algorithm solves the all-pairs shortest-paths problem on a directed graph. Bellman-Ford solves for the node values relative to a "zero reference" node, then reverses the arcs (thus making it a single-sink problem), and solves again, thus we tighten the event values, but not the constraint values. Therein lies the difference between Floyd-Warshall and Bellman-Ford as far as this paradigm. We assume specified constraints, in general, should only be changed by the knowledgebase user.

Given weighted, directed graph $G = (V, E)$, the Bellman-Ford algorithm uses relaxation, decreasing the estimated vertex weight $d[v]$ of the shortest path between the source node (zero reference) $s$ and each vertex $v \in V$ until the actual shortest-path weight $\delta(s, v)$ is achieved [14]. If a negative-weight cycle is obtained during computation, the graph is inconsistent.
The algorithm, as it appears in Cormen, *et. al.* [14] is outlined in Figure 9.3.

Bellman-Ford$(G, w, s)$

1. for each vertex $v \in V[G]$
   
2. do $d[v] \leftarrow \infty$; $\pi[v] \leftarrow nil$

3. $d[s] \leftarrow 0$

4. for $i \leftarrow 1$ to $|V[G]| - 1$

5. do for each edge $(u, v) \in E[G]$

6. if $d[v] > d[u] + w(u, v)$

7. then $d[v] \leftarrow d[u] + w(u, v)$; $\pi[v] \leftarrow u$

8. for each edge $(u,v) \in E[G]$

9. do if $d[v] > d[u] + w(u, v)$

10. then return INCONSISTENT

11. return CONSISTENT

Figure 9.3: The Bellman-Ford single-source shortest-paths algorithm.

The for loop in lines 4 – 7 relaxes each edge. This is performed $|V| - 1$ times. Lines 8 – 11 then verify that no negative cycle has been created. The algorithm thus runs in $O(VE)$ time; the initialization being linear in the number of vertices ($\Theta(V)$), and each traversal of all edges ($O(E)$) being performed $|V| - 1$ times. A non-asymptotic performance improvement can be made by checking for negative cycles in the edge traversal, short-circuiting the algorithm as soon as inconsistency is becomes evident.

The Bellman-Ford algorithm assumes point-weights on the nodes and edge-weights. It is readily adapted, however to an interval model, where the interval weights from the source node correspond to early and late (uncertain) values on a node or edge. Our tuple model can be decomposed to an interval model by splitting each sixtuple event into two nodes (start and finish) at the time the algorithm is applied.

Figure 9.4 shows the mapping of an event in 6-tuple format to the interval equivalent. Note duration now expresses an interval constraint between start and finish time. The 8-tuple constraint between two events is translated to interval constraints as shown in Figure 9.5.

As mentioned above, the need to specify non-convex constraints between events arises frequently in practice. For this work, we have applied a heuristic called Path Consistency to reduce convexity in a network prior to solution. We then solve the network, choosing one constraint in turn on each disjoint constraint; thus evaluating the Cartesian product of all constraints. Consistent solutions are presented to the user one at a time, allowing the “next” consistent solution to be sought out and displayed.
Figure 9.4: The sixtuple event model and interval equivalent

Figure 9.5: Two sixtuple events, constrained by an eighttuple constraint
9.4 Path Consistency

The constraint satisfaction heuristic of path consistency operates by imposing local consistency on the variables of the graph. Path consistency was first used by Montanari [20], and has been further explored by Dechter et. al. [16], VanBeek & Cohen [24], among others. Path consistency can be used to assist consistency checking. It is necessary, but not sufficient for consistency verification. Thus, it is used as a heuristic to simplify the problem before another method, e.g., Bellman-Ford is used. If it detects inconsistency, however, we know the network to be inconsistent.

Path consistency operates by examining a fully-connected version of the graph in question. Unspecified constraints from the original graph are initialized as "unconstrained." The algorithm then tightens the network, considering the constraints and event-variables in triples. A constraint-path of length two on this triple of events is compared to the third constraint. If this third constraint is contained in (a subset of) the pair of constraints, it is updated to their net effect. When the edges stop changing, we have reached path consistency. The network can be found to be inconsistent during this process if an edge ever has no constraints which are valid. When exclusively qualitative constraints are used, the path consistency algorithm runs in $O(n^3)$ time, where $n$ is the number of events.

An algorithm for computation of path consistency is outlined in Figure 9.6. Graph $G$ is assumed to be a working copy of the original $n$-node graph, over which we wish to compute path consistency. The list of (disjoint) constraints on edge between nodes $i$ and $j$ is denoted $(i, j)$.

A more efficient version of path consistency keeps queues of edges which need to be re-examined in the next iteration, thereby reducing the operations required each pass through the network. When this list is exhausted, the procedure is completed. This procedure is described by Dechter et. al. [16].

The network is inconsistent if we find an edge for which no constraint is consistent in terms of its neighbors. Complex networks can be checked for inconsistency before the full consistency check and propagation.

There are three basic operations necessary to implement path consistency: intersection, composition, and smoothing. The data structure used for doing operations on disjunctions of intervals is a pair of sorted, linked-lists. One list holds the start points, while the other holds the finish points of the intervals on the edge.

Intersection between two sets of intervals, $I_1$ & $I_2$, admits only values which are in both. Intersection is denoted:

$$I_1 \oplus I_2$$

To illustrate, consider:

$$I_1 \leftarrow \{(1, 3), (4, 6)\}$$
$$I_2 \leftarrow \{(0, 1), (2, 5)\}$$

$$I_1 \oplus I_2 = \{(1, 1), (2, 3), (4, 5)\}$$
Composition of two sets of intervals takes a complete pairwise mapping of start and finish times so that the composition, $C$ of $I_1$ and $I_2$ is denoted:

$$C = I_1 \otimes I_2$$

And calculated for each interval in each (all-pairs):

$$c(\text{start}) = i_1(\text{start}) + i_2(\text{start})$$

$$c(\text{finish}) = i_1(\text{finish}) + i_2(\text{finish})$$

where:

$$c(\text{start}) \in C(\text{start}) \text{ and } c(\text{finish}) \in C(\text{finish})$$

$$i_1(\text{start}) \in I_1(\text{start}) \text{ and } i_2(\text{start}) \in I_2$$

$$i_1(\text{finish}) \in I_1(\text{finish}) \text{ and } i_2(\text{finish}) \in I_2$$

For example, if we have:

$$I_1 \leftarrow \{(2,3),(6,8)\}$$

$$I_2 \leftarrow \{(0,1),(2,2)\}$$

$$I_1 \otimes I_2 = \{(2+0,3+1),(2+2,3+2),(6+0,8+1),(6+2,8+2)\}$$

which simplifies to:

$$I_1 \otimes I_2 = \{(2,4),(4,5),(6,9),(8,10)\}$$

Note that some of the intervals in this solution overlap. The sorted linked-list data structure for the disjunction of intervals assumes that the intervals do not overlap or contain subintervals. To minimize "redundant" data like this, we must "smooth" the interval list. This must be performed after composition, which is prone to creating redundant interval information.

Smoothing an interval is performed by sweeping through the values on the interval, tracking the number of "active" intervals, and removing unneeded starts and finishes.

From our composition example:

$$I_1 \otimes I_2 = \{(2,4),(4,5),(6,9),(8,10)\}$$

when the solution set of intervals is smoothed:

$$I_1 \otimes I_2 = \{(2,5),(6,10)\}$$

Intersection can be implemented in $O(n)$, where $n$ is the number of disjunctions, by using an insertion sort (they are already ordered). Composition requires a Cartesian product, thus it requires $O(n \times m)$, where $m$ and $n$ are the number of intervals in each list, respectively. Smoothing takes $O(n)$ time where $n$ is the number of intervals in the list to be smoothed.
9.5 Interval Trees

To perform efficient queries such as “Which events are potentially active between the hours of 10am and 6pm?” or “What events must be active on February 7th?”, two features must be added to the model.

The first feature is a calculation on the sixtuples to provide an interval over which the event is possibly occurring or known to be occurring. The former is trivial, being nothing more than the early-start and late-finish times on the event. The latter is a bit more complex; the calculation of known time is shown in figure 9.7.

We define the known time as the interval over which the event must occur

\[ [I_{\text{start}}(\text{known}), I_{\text{finish}}(\text{known})]. \]

The interval (LateStart, EarlyFinish) is a naive answer, but we find that duration semantics must be taken into consideration, especially if you recognize that events may not be fully specified (e.g. they might just have early start and duration data). Note that there may not even be a time at which the event must occur, if it is underconstrained. The above formula will choose the more constraining (conservative) time, if the finish of this interval occurs before the start, then we cannot specify a known time for the given event. “Tightening” of the sixtuples in question must be performed prior to the calculation of the known interval, to insure values are in line with one another. The method for tightening is shown in figure 9.8.

The second feature we must add for efficient queries is a data structure for storage and retrieval of these possible/known intervals. An interval tree [21] allows performance of storage, and interval and point queries over intervals in optimal time\(^1\). This feature is particularly useful when large systems of databases of events are used.

Interval trees consist of a primary, static skeleton that is a balanced, binary tree whose in-order traversal yields the sorted list of endpoints of the interval set. A secondary overlay indicates active sub-branches, thereby pruning the tree during search. Non-leaf nodes correspond to the interval over which their child-nodes are active. Intervals are uniquely stored at the highest level fully containing the endpoints. Detailed discussion of this structure and its use can be found in [21].

9.6 The Tachyon User Interface

9.6.1 Objectives

The Tachyon system we developed was first embedded into the Patti\(^{++}\) software, so that we could gather information for verification and performance tests. For special tests on temporal examples, we encapsulated this code into a “batch mode” form by adding some routines to perform I/O on simple text files. To market the system beyond performing benchmarks and technical demonstrations, we built a graphical user interface (GUI). A picture of the Tachyon interface with a sample network is shown in Figure 9.9.

\[^1\text{Intersection can be found in optimal time } \Theta(N \log(N)) \text{ with } O(N \log N) \text{ preprocessing time and using optimal space } O(N) \text{ Insertions and deletions can be performed in } O(\log(N)) \text{ time.}\]
This graphical editor allows loading and saving of temporal data files that are compatible with the batch mode version. Thus, one could use Tachyon to test consistency of a network, fine tune it, or test it in "What if..." scenarios as desired, then save the network in a file the batch-mode system can use. The interface itself is a CAD-like direct-manipulation editor for the graphical representation of the underlying network. Popup panels allow manipulation of the data within the network.

The architecture of the Tachyon interface is illustrated in Figure 9.10. The GUI allows intimate interaction between the user and a network. The user can make incremental changes and immediately observe their side-effects. The core reasoning-engine remains capable of running as an embedded process or as a batch job through pipes or files. We are also isolating the interface itself to provide a generic graph editor library for future applications.

9.6.2 Functionality

Canvas Operations

The canvas allows the user to layout the network graphically. Nodes appear as boxes with name and tuple information within. Edges (optionally labeled) appear as lines between the nodes. Direction of the constraint is indicated by an arrowhead. Nodes may be created so long as they do not overlap another. They may be moved as desired (moving their edges with them), as long as they do not overlap another node at the final "drop" location. We disallow overlap to avoid ambiguity when picking objects. Edges may only be created between two existing node, and can only be moved by moving the nodes.

Objects (nodes and edges) may be selected by left-clicking the mouse over them. This will unselect all other objects unless the user shift-clicks to select. Multiple objects can also be selected by enclosing the desired objects in a rectangle drawn by right-dragging. Selected objects can be deleted by clicking the Delete icon. If exactly one object is selected, clicking on the Edit icon will bring up the I/O panel for that object (see next section). This panel may also be brought up by double-clicking on the desired object.

Meta-left on an object will pop-up a brief, descriptive window (a "Peek") of the object's data. This window will pop-down as soon as the mouse button is released.

System and User I/O

The Node Information menu (figure 9.11) is pretty straightforward, allowing direct changes to be made to the name-tag, temporal values of the event associated with the node, and verbose description of the event.

In contrast, the Edge Information menu is very complex. The default representation for a temporal distance is an 8-tuple, as described earlier, and shown in the top portion of figure 9.12. Allen relations are qualitative relationships between events. Each Allen relation has a corresponding 8-tuple, which the system substitutes for it at solution time. For example, "before" would be represented by a minimum distance of $\epsilon$ (epsilon) and maximum distance of $+\infty$ (positive infinity) on all four of the 8-tuple pairs. This says that a non-zero positive (but otherwise unknown) distance exists between the start and finish times of event 1 and the start and finish times of event 2.
The Allen button on the menu toggles between allowing entry into the 8-tuple values and allowing selection of Allen relations. Selection of Allen relations will disable other Allen relations in order to force convexity of the given distance. In the example in figure 9.12, Before and Meets are selected, disabling all but Before, Meets and Overlaps. Parameters for the Allen relations might be altered and/or disabled based on selection and deselection of other relations. If Meets is selected, for instance, the minimum values of Before and Overlaps are set to e, to maintain convexity.

Nonconvexity on an edge is entered by selecting the Add button after entering the first relationship. This produces a new set of tuples and Allen relations, which may be used in any combination on nonconvex edges. Paging between multiple relations on a single edge is accomplished by the buttons with the left and right arrow on them (← & →). Displayed between these are the index and total number of relations on the edge (e.g., “1 of 4”). The Annotation area at the bottom of the menu contains a text region and a “cycle button”. The options on this button select whether the edge will be: unlabeled, have a user-specified label, or be labeled according to the constraint(s) on the edge.

The information from the currently-displayed distance can be inserted into the text region by clicking the “Append Distance Information to Annotations” button. This region supports many of the emacs editing functions.

There are two files associated with networks created in the editor. The first is identical to the files used by the batch-mode version. The second file specifies layout information, modes, etc. We intend these two files to merge in the future. This includes being able to layout a graphic view of the network from just the constraint information.

Network Operations

The user can “lock” the current values the event sixtuples are instantiated to and “revert” to these variables when desired. This facilitates incremental fine-tuning of a system. The current instantiations can also be compared to the locked values to determine which parts of the graph have changed.

There are two ways to ask the system to solve the current network. The first always solves the network using the current sixtuples and active convex constraints (if disjoint). Networks with disjoint constraints may have multiple consistent solutions, so a second way to solve the network is to reset all the active constraints and start from the default values. This will search for the first configuration of the disjoint constraints that produce a consistent solution. The active constraint on each edge will be indicated. The current “state” of the network is saved so we can solve for the “next” consistent solution.

Time Line

Gantt charts are a common presentation format for project management and scheduling. Tachyon is capable of creating a view akin to a Gantt chart, based on the current tuple values on each node. An example of this is shown in Figure 9.13.

The events are represented by blocks such as the one shown in Figure 9.14. Each is appropriately positioned on the time line and stacked vertically. The length of the block corresponds to the possible length of the event. The left side corresponds to the earliest
possible start time, while the right side corresponds to the latest possible finish time. The interval of uncertainty for the start time is represented by a green band in the top left corner. Similarly, a red band in the bottom right corner indicates the interval of potential finish times. The length of the two black bands across the center of the block show the minimum and maximum durations for the event. Infinities in all cases are drawn to the left and/or right side of the screen, as appropriate.

Our experience has been this representation is quite intuitive, requiring minimal explanation, and unambiguous. Thus, we are able to meaningfully display the events on a time line, without losing information.

When this screen is invoked, an interval tree is created. The screen also allows point and interval queries to be made on the events in the time line. The events responding to the queries will invoke a screen of their own, providing the desired "slice" of the events to be examined. This is performed by Meta+Middle-mouse. A simple click will perform a point query across the events, while dragging an interval will perform an interval query. The subsequent window will have all the functionality of the original.

We also have enabled this screen to choose crisp intervals for events, which are then copied back to the network. The intent is events which have occurred in a project (or whatever) can be explicitly set to their actual (certain) values, and incorporated into the model (allowing subsequent projection, etc.)
boolean procedure PATH\_CONSISTENCY

1. Fully connect graph $G$ by adding unconstrained edges
2. still\_updating ← TRUE
3. while still\_updating the matrix
4. still\_updating ← FALSE
5. for $k \leftarrow 1$ to $n$ do
6. for $j \leftarrow 1$ to $n$ do
7. for $i \leftarrow 1$ to $n$ do
8. $\text{temp} \leftarrow (i, j) \oplus (i, k) \otimes (k, j)$
9. if $|(i, j)|$ is empty
10. return INCONSISTENT
11. if $\text{temp}$ differs from $(i, j)$
12. still\_updating = TRUE
13. return CONSISTENT

Figure 9.6: Path consistency algorithm

function KNOWN

1. TIGHTEN the event sixtuple
2. $I_{\text{start}}(\text{known}) \leftarrow \min((\text{Late}\_\text{Finish} - \text{Minimum}\_\text{Duration}), \text{Late}\_\text{Start})$
3. $I_{\text{finish}}(\text{known}) \leftarrow \max((\text{Early}\_\text{Start} + \text{Minimum}\_\text{Duration}), \text{Early}\_\text{Finish})$
4. if ($I_{\text{finish}}(\text{known}) < I_{\text{start}}(\text{known})$) then
5. there is no known time

Figure 9.7: Calculation of an event’s known interval
procedure TIGHTEN

1. if \( (M_{\text{in\_duration}} < E_{\text{arly\_finish}} - L_{\text{ate\_start}}) \)
2. \( M_{\text{in\_duration}} \leftarrow E_{\text{arly\_finish}} - L_{\text{ate\_start}} \)
3. if \( (M_{\text{ax\_duration}} > L_{\text{ate\_finish}} - E_{\text{arly\_start}}) \)
4. \( M_{\text{ax\_duration}} > L_{\text{ate\_finish}} - E_{\text{arly\_start}} \)
5. if \( (E_{\text{arly\_finish}} - M_{\text{ax\_duration}} > E_{\text{arly\_start}}) \oplus (E_{\text{arly\_start}} + M_{\text{in\_duration}} > E_{\text{arly\_finish}}) \)
6. \( E_{\text{arly\_start}} \leftarrow E_{\text{arly\_finish}} - M_{\text{ax\_duration}} \)
7. or
8. \( E_{\text{arly\_finish}} \leftarrow E_{\text{arly\_start}} + M_{\text{in\_duration}} \)
9. if \( (L_{\text{ate\_start}} + M_{\text{ax\_duration}} < L_{\text{ate\_finish}}) \oplus (L_{\text{ate\_finish}} - M_{\text{in\_duration}} < L_{\text{ate\_start}}) \)
10. \( L_{\text{ate\_finish}} \leftarrow L_{\text{ate\_start}} + M_{\text{ax\_duration}} \)
11. or
12. \( L_{\text{ate\_start}} \leftarrow L_{\text{ate\_finish}} - M_{\text{in\_duration}} \)

Figure 9.8: Tighten the sixtupple so that start and finish are consistent with duration
Figure 9.9: The Tachyon temporal constraint network editor.

Figure 9.10: System architecture for TCN / Tachyon
Figure 9.11: Node Information menu

Figure 9.12: Edge Information menu
Figure 9.13: A time line view of the lunch example (instantiated)

Figure 9.14: Event representation on the time line.
Chapter 10

Discussion & Related Issues

We have described a new temporal model which combines an extremely expressive model with high-performance computation techniques. Our model is the first to allow expression of uncertainty and duration in a single event, qualitative, parameterized qualitative, and quantitative events, and calculation over disjoint constraints. It holds a great deal of promise for further exploration of heuristic techniques of dealing with the intractability of reasoning over disjoint constraints.

An unexpected issue encountered while addressing the practical applications of the model was that of enumerating time itself. Different granularities are desired for different applications. Project management is most likely to want times expressed in days and dates, whereas a factory process scheduler would use minutes. If the latter were a subnetwork of the former, the two would have to be able to merge their two networks somehow.

To accomplish this, a special type was created. This type internally stored everything in seconds, but would input and output itself in terms of what sort of granularity the user was interested in. This problem is trivial when nothing more than weeks, days, hours, minutes and seconds are used, but adding months, years, and dates introduces some complexity.

To overcome this, we took advantage of the time functions built-in to C/Unix which provide such a translation capability. This had the disadvantage of limiting the universe to the era of Dec. 13th, 1001 to Jan 18th, 2038. This is caused by the 32-bit limit on the long int type in C. The system stores the number of seconds since Jan. 1st, 1970. Thus, given it's finite continuum, we have a limitation on the period over which this solution is valid. We do not plan to address this problem ourselves immediately, in hopes that later versions of the UNIX system people will provide a solution themselves.

A side effect of this limitation is that when we express unconstrained events, we must differentiate ±∞ from the limit dates. To make things worse, since the model is numeric, rather than symbolic, it is possible to appreciably decrement from infinity. To keep infinity infinite throughout the calculations special care is taken by the algorithms and new time type. This is provided at slight performance cost.
Chapter 11

Future Directions and Conclusions

We have described a model for temporal reasoning and a corresponding environment providing opportunity for evaluation and experimentation. The model offers considerable expressiveness without severe performance drawbacks. We are still exploring the performance characteristics. Some areas where we see potential for use of this model include: scheduling satellite use, project planning, equipment delivery/deployment, job scheduling in manufacturing, and temporal consistency checks for knowledge bases.

Tachyon is a prototype tool, and as such we are constantly modifying it. There are many enhancements we are adding to the Tachyon GUI, including: hierarchical representation, PostScript output, panning, zooming, true date representation capabilities, and some cosmetic enhancements. By “hierarchical,” we mean an entire (sub)graph may be represented by a single node in a view of the graph at a more abstract level. We believe hierarchical capabilities are necessary to process large graphs (e.g., 10,000 nodes) in a form meaningful to people, and we are investigating the issues involved.

We have assumed the small problems can be solved quickly, and their dependencies on other components can be resolved reasonably quickly. We are currently exploring the applicability of several graph-based decomposition techniques to bringing down the cost of searching for a feasible solution to problems in which there is a nontrivial number of non-convex constraints present after simple heuristics, e.g., path consistency, have been applied. For instance, when the edges of a graph with disjunctive constraints (e.g., “before or after”) form planar subgraphs; algorithms, such as the Planar Separator Algorithm, developed by Lipton and Tarjan [19], can be exploited to handle a subset of the intractable problems. This algorithm uses decomposition to simplify the problem by dividing the network into small parts with minimal dependency on one another. The decomposition method is “divide-and-conquer,” breaking the problem into multiple smaller problems, that are recursively decomposed. The Planar Separator Algorithm tries to divide the problem into roughly equal subparts, by estimating component costs based on vertex weights.

There are many features we foresee as beneficial long-term goals. We have already adopted a CAD-like interface, and adding the ability to express entire networks as a single event in a higher-level network (hierarchically) would greatly enhance visualization and possibly reduce network complexity.

Right now the model chooses between disjoint constraints arbitrarily. We would like to provide some preference-specification capabilities to assist in finding a solution better fitting
the desires of the user.

As noted earlier, we do not attempt to backtrace the cause of inconsistency. This makes it difficult to debug a complex, inconsistent network. Providing diagnostic information to identify and remedy constraint failure would address this problem.

An issue often intimately related to temporal reasoning in the scheduling and planning arenas is that of resource management. Time is essentially a resource to be allocated as needed, and other resources require consideration in the context of the schedule. Some resources are expendable, e.g., we have 20 gallons of red paint, while others are renewable, but limited in number, e.g., we have 3 lathes, each capable of performing a single job at a time.
Chapter 12

Project Planning Example

We will illustrate a common use for temporal representation and reasoning by showing a generic Software Engineering example. Figure 12.1 shows a generic template for a constraint network with three initial values specified: Early and Late Start (August 4th - 6th) for the Customer Meetings and Late Finish (October 30th) for the Spec Approval. To see temporal windows for the events between, we propagate and get the network as shown in Figure 12.2.

A library of templated for various projects commonly performed could be archived and retrieved when a new operation of the same type is performed. Prior solutions to similar problems could also be archived for comparison to present situations.
Figure 12.1: Template for Software Engineering

Figure 12.2: Software Engineering example, propagated
Chapter 13

Scheduling Example

Our second example involves scheduling use of limited resources by multiple clients. We have four newspapers, the Guardian, the Daily Express, the Financial Times, and the Star. Four people read these papers each morning. Each takes a different amount of time to read each paper. Exactly one person can read a single paper at a time, and each reader has an order in which they prefer to read the papers.

By expressing "before or after" constraints between the events corresponding to a particular person reading a particular paper, we can have Tachyon choose an ordering for the paper circulation, constrained by the various start times and the desired completion time (for lunch). This example is taken from [17].
Figure 13.1: Template for Scheduling Paper Use

Figure 13.2: Scheduling Example, propagated
Part III

Integration of Case Based Reasoning and Temporal Reasoning
Abstract

We describe the integration of two prototype software tools currently under development at GE-CR&D: CAFE, a case-based tool for expansion of forces, and Tachyon, a tool for constraint-based temporal reasoning. The goal of the integration is to provide operational users with the ability to custom tailor forces for a current mission by drawing from historical cases, at the same time tracking the effect of temporal constraints on those forces through instantiation and deployment, thus facilitating faster, better force development and deployment.
Chapter 14

Background

14.1 CAFS/CAFE

Case based reasoning (CBR) involves solving new problems by identifying and adapting similar problems stored in a library of past experiences/problems. CBR systems are comprised of a case-library, indexing, matching and retrieval mechanisms, and a reasoning component. The important steps in the inference cycle of CBR are to find and retrieve cases from the case library which are most relevant to the problem at hand (probe) and to adapt the retrieved cases to the current input. The matching and retrieval mechanisms, driven by the current context (reasoner’s goal and probe), return the most similar cases from the case memory. Similarity among cases is based on an evaluation of salient and relevant features [57]. The reasoning component processes the retrieved cases, adapting their solutions (plans, explanations, interpretations) to apply in the current situation.

CAFS is a Case-Based Reasoner designed to select forces for military missions. Currently, CAFS receives probes from SRI’s planning system SOCAP, which consist of information on a military task, its location, and the expected threat at that location. CAFS returns to SOCAP the available force(s) best suited to successfully completing that task.

Features used in case indexing, retrieval and matching include:

- the type of task (e.g., set-up ground-defense or establish evacuation center),
- the terrain at the location, and
- the type of threat (e.g., terrorist cell or volcano eruption).

Figure 14.1 illustrates a fragment of the mission hierarchy used by CAFS to identify the probe’s tasks and guide the matching of the case’s tasks. A simple semantic distance measure is used to compute similarity. A force hierarchy, similar to the mission hierarchy, completes the taxonomical knowledge used by the CBR. The case library contains the CBR episodic knowledge [3].

Additional features can be added by users of the system as their usefulness is established (e.g., the climate of the region or the expected weather). Once a set of possible matching cases is retrieved from the library, CAFS develops a set of force suggestions based on the retrieved cases solutions. For those cases where there is an exact match, CAFS attempts to find an
available force of the same type used in the retrieved case. If such a force is unavailable, CAFS attempts (through adaptation) to find an available force that is similar enough to the retrieved solution that it also could successfully complete the task. This adaptation is based on the forces type (i.e., infantry unit or medical evac unit) and capabilities. When a retrieved case is not an exact match, CAFS first tries to adapt the required capabilities from the retrieved case using the differences found between the probe and the retrieved case. Then, starting with the solution from the retrieved case, CAFS attempts to find (using adaptation) an available force that has the required capabilities. Once a plan as been completed (using SOCAP, in a recent application), new force selection cases can be extracted from the plan and added to the case library for future use. Having completed our Technology Integration Experiment (TIE) with SOCAP, we plan to extend the same CAFS capabilities to TARGET, a collaborative mixed initiative planning environment currently under development by BBN.

Figure 14.2 illustrates the matching and retrieval process for a ground patrol task. Three force modules have been retrieved and partially ordered according to an aggregate measure of match to mission requirements. The top-ranked solution is displayed in the top right corner of the figure.

CAFE takes the major force list generated by a planning system, e.g., SOCAP, during course of action (COA) development phase and return a complete set of forces (both combat and support forces) appropriate to the plan (based on missions, location, weather, etc.).

This expansion is done by retrieving previous cases and adapting the expanded force list from best matching previous case. When an appropriate case cannot be found, a generic expanded force can be generated using rules (like those in the Automatic Force Generation Package) or component information from the force module database. If tailoring information can be retrieved from current planners, these generic forces can then be specialized to the context of the current case. The set of forces output from the CAFE can then be analyzed.
**Figure 14.2:** Probe description and retrieved force in CAFS.
for supply and resupply needs, scheduling choices, etc.

14.2 Tachyon

Tachyon is a constraint-based model for representing and reasoning about both qualitative and quantitative aspects of time, together with a software implementation of that model. Temporal reasoning problems arise in numerous computer applications: databases, simulators, expert systems, and industrial scheduling and planning systems (minimizing assembly line slack time, projecting critical steps in a deployment plan to insure proper interaction between them, etc.) all need to manipulate temporal information to model the world. In developing both Tachyon's data model and software implementation (our current software prototype is implemented in C++ using X-Windows and extensions to the InterViews class library, and is compiled for the Sun Sparcrstation), we have tried to provide the versatility and power to handle effectively a variety of temporal reasoning problems typically arising in planning and scheduling applications, in keeping with our goal of producing a powerful and versatile tool. Some of the key features we provide are listed below:

- deal with uncertainty regarding the exact time and duration of occurrence of events\(^1\), e.g., X will occur sometime in the morning, and refueling takes between 15 and 40 minutes,
- express both quantitative and qualitative constraints between events, e.g., X is before or meets Y, and X ends between 10 and 15 minutes before Y starts,
- express parameterized qualitative constraints between events, e.g., X is before Y by at most 6 days,
- provide multiple granularities, e.g., seconds, hours, days, etc., and their combinations, e.g., days:hours:minutes, day:month:year,
- promote ease of use via graphical input and display capabilities,
- run as a subprocess in other applications as well as stand-alone,
- utilize techniques that will remain effective even in very large application domains,
- serve as a versatile testbed for exploring new techniques for coping with the intractability associated with disjoint constraints.

One of the key reasons we began developing Tachyon was as a research vehicle to explore new techniques for dealing with the inherent complexity of temporal reasoning and scheduling. We recognized it's applicability to a number of problems of both military and commercial interest, and have simultaneously sought opportunities to explore the appropriateness of using Tachyon in a diverse set of applications. In addition to the prototype integrating

\(^1\)Although we will use the term event, it should be noted that one could as easily refer to an arbitrary proposition that has temporal extent.
Tachyon with CAFE described in this paper, we have applied it to plan recognition tasks, where it was used to validate temporal sequencing of events as an aid in formulating plan hypotheses, to plan generation and monitoring, to scheduling for plastics and power systems manufacturing, and to retrieval and situation refinement in a prototype spatio-temporal data management system. In this last application, we used Tachyon’s constraint propagation capabilities together with partial information about interrelated events to provide intra-force temporal refinement for tasking support [5, 7].

The interested reader is referred to [1, 6] for more technical details on Tachyon.
Chapter 15

Integrated Capabilities

One important aspect of force expansion is incorporating all the information from the major force list into the full force list. A good example of this is the required delivery date (RDD) that the planner associates with each major force based on the COA. As each major force is expanded into its component units, and non-organic support forces are added in response to projected needs of the force, the RDD (as well as other major force level information) must be passed down to the lower level units. This is not simply a direct translation. Temporal constraints exist between the units of a force as well as between major forces (for example, the unloading crews for an airfield must arrive before the cargo planes). CAFE represents the explicit temporal constraints in such a way that Tachyon can be used to check for temporal consistency over the entire force, and subsequently to maintain maximal regions of temporal feasibility for the forces as the COA evolves. The addition of these explicit temporal constraints will allow greatly expanded flexibility in adapting the time phasing of a force to the resource constraints which exist at the time of plan execution.

15.1 An Example

An simple example of the integrated use of Tachyon and CAFE for force expansion and time phasing is given in Figures 15.1 and 15.2 below. While the prototype can handle arbitrary temporal constraints, for simplicity we will use only a simple restriction on RDDs. In Figure 15.1, the user has selected a force for expansion and that force has been expanded into its component units and non-organic support forces, which are listed in the highlighted section of the CAFE window (top of figure). Note that the RDDs of the component forces are not known at this time.

Once the expansion of the major force has been determined, we can, from the associated temporal constraints, use Tachyon to compute the modified RDDs for the expanded force, as shown in Figure 15.2.

In order to implement this temporal reasoning aspect of the Case Based Force Expansion Module we need to capture the temporal constraints which hold between the components of a force. The inter force constraints will be based on the COA and coded into the plan, e.g., by SOCAP operators, but the intra force constraints are expected to be plan independent, although they may be mission or location dependent. We plan to obtain intra force
Figure 15.1: CAFE (top) and Tachyon (bottom) in coordinated use
Figure 15.2: After Tachyon has propagated the stated temporal constraints and returned the modified RDDs to CAFE.
constraints from the same sources who will provide us with the knowledge about force structure. By integrating Tachyon into a force module editor, we will be able to capture temporal constraints as force modules are acquired.
Chapter 16
Future Directions & Conclusions

16.1 ForMAT Integration

One of the key requirements for making the integrated Tachyon/CAFE tool succeed is that we must capture cases and temporal constraints in force libraries. Little of this information has been captured to date, largely because there has been no technology available to support its exploitation. As tools begin to emerge that can exploit the data, a parallel need emerges for tools to capture it. We have recently begun an effort to develop a Force Module Editor for creating and modifying force modules. This effort will integrate ForMAT (a Force module/TPFDD editor being developed by MITRE), Tachyon, and the FM and Mission ontologies developed for CAFS. This Force Module Editor will also be used to capture the temporal constraints that hold among force components. This will provide a knowledge acquisition tool for information about force structure and force usage, and a smart editor for modifying existing force modules.

We will also integrate CAFS matching and ranking capabilities with ForMAT. As a result of this integration, the force modules retrieved by ForMAT will be (partially) ranked according to their degrees of matching with the mission requirements specified by the probe. The user will be able to analyze the results, observe the difference in force capabilities among cases that lead to different partial matches, and express his/her preference by changing the saliency of the features used in the matching process.

16.2 Extended Capabilities of Tachyon

There are also several planned extensions to Tachyon that will enhance its ability to be used effectively when deployed in an integrated framework such as that described above. These include hierarchical representations, which will allow a user to work with temporal constraints on forces at an arbitrary level of detail without direct concern for constraints at lower levels, and a greatly expanded “debugging” capability designed to provide non-expert users with the ability to recover from situations of temporal inconsistency.
16.3 Conclusions

We have demonstrated the integration of two technologies to provide a powerful tool for developing and maintaining forces for crisis response. The integrated CAFE/Tachyon prototype has now been demonstrated to several groups of domain experts, and has been well received by that community. The biggest and most immediate obstacle to fielding this capability is that no one has captured the force packages or their associated intra force constraints in a disciplined way to date. Our planned work to integrate the prototype with Mitre's ForMAT tool should help to minimize this obstacle. We plan to be able to demonstrate an integrated ForMAT/CAFE/Tachyon prototype later this year.
Bibliography


<table>
<thead>
<tr>
<th>addresses</th>
<th>number of copies</th>
</tr>
</thead>
<tbody>
<tr>
<td>DONALD F. ROBERTS</td>
<td>5</td>
</tr>
<tr>
<td>ROME LABORATORY/C3CA</td>
<td></td>
</tr>
<tr>
<td>52 BROOKS ROAD</td>
<td></td>
</tr>
<tr>
<td>GRIFFISS AFB NY 1341-4505</td>
<td></td>
</tr>
<tr>
<td>GENERAL ELECTRIC CORPORATE RESEARCH</td>
<td>5</td>
</tr>
<tr>
<td>1 RIVER ROAD</td>
<td></td>
</tr>
<tr>
<td>SCHEMENCTADY NY 12309</td>
<td></td>
</tr>
<tr>
<td>RL/SUL</td>
<td>1</td>
</tr>
<tr>
<td>TECHNICAL LIBRARY</td>
<td></td>
</tr>
<tr>
<td>26 ELECTRONIC PKY</td>
<td></td>
</tr>
<tr>
<td>GRIFFISS AFB NY 13441-4514</td>
<td></td>
</tr>
<tr>
<td>ADMINISTRATOR</td>
<td>2</td>
</tr>
<tr>
<td>DEFENSE TECHNICAL INFO CENTER</td>
<td></td>
</tr>
<tr>
<td>DTIC-FDAC</td>
<td></td>
</tr>
<tr>
<td>CAMERON STATION BUILDING 5</td>
<td></td>
</tr>
<tr>
<td>ALEXANDRIA VA 22304-6145</td>
<td></td>
</tr>
<tr>
<td>ADVANCED RESEARCH PROJECTS AGENCY</td>
<td>1</td>
</tr>
<tr>
<td>3701 NORTH FAIRFAX DRIVE</td>
<td></td>
</tr>
<tr>
<td>ARLINGTON VA 22203-1714</td>
<td></td>
</tr>
<tr>
<td>NAVAL WARFARE ASSESSMENT CENTER</td>
<td>1</td>
</tr>
<tr>
<td>GIDEOP OPERATIONS CENTER/CODE QA-50</td>
<td></td>
</tr>
<tr>
<td>ATTN:  E RICHARDS</td>
<td></td>
</tr>
<tr>
<td>CORONA CA 91719-5000</td>
<td></td>
</tr>
<tr>
<td>WRIGHT LABORATORY/AAAII-2</td>
<td>1</td>
</tr>
<tr>
<td>ATTN:  MR FRANKLIN HUTSON</td>
<td></td>
</tr>
<tr>
<td>WRIGHT-PATTERSON AFB OH 45433-6543</td>
<td></td>
</tr>
<tr>
<td>AFIT/LDDE</td>
<td>1</td>
</tr>
<tr>
<td>2950 P STREET</td>
<td></td>
</tr>
<tr>
<td>WRIGHT-PATTERSON AFB OH 45433-6577</td>
<td></td>
</tr>
</tbody>
</table>

DL-1
PAUL R. COHEN
UNIV OF MASSACHUSETTS
COINS DEPT
LEDGERE GRC
AMHERST MA 01003

MS. MARIE DEJARDINS
SRI INTERNATIONAL
333 RAVENSWOOD AVENUE
MENLO PRK CA 94025

JON DOYLE
LABORATORY FOR COMPUTER SCIENCE
MASS INSTITUTE OF TECHNOLOGY
545 TECHNOLOGY SQUARE
CAMBRIDGE MA 02139

DR. BRIAN DRAPPLE
AI APPLICATIONS INSTITUTE
UNIV OF EDINBURGH/80 S. BRIDGE
EDINBURGH EH1 LHN
UNITED KINGDOM

MR. SCOTT FOUSE
ISX CORPORATION
4353 PARK TERRACE DRIVE
WESTLAKE VILLAGE CA 91361

MR. STU DRAPER
MITRE
EAGLE CENTER 3, SUITE 8
O'FALLON IL 62269

MARK FOX
DEPT OF INDUSTRIAL ENGRG
UNIV OF TORONTO
4 TADDLE CREEK ROAD
TORONTO, ONTARIO, CANADA

MR. GARY EDWARDS
4353 PARK TERRACE DRIVE
WESTLAKE VILLAGE CA 91361

MS. MARTHA FARINACCI
MITRE
7525 COLSHIRE DRIVE
MCLEAN VA 22101
MR. RUSS FREW
GENERAL ELECTRIC
MOORESTOWN CORPORATE CENTER
BLDG ATK 145-2
MOORESTOWN NJ 08057

MICHAEL FEHLING
STANFORD UNIVERSITY
ENGINEERING ECO SYSTEMS
STANFORD CA 94305

MR. RICH FRITZSON
CENTER FOR ADVANCED INFO TECHNOLOGY
UNISYS
P.O. BOX 517
PAOLI PA 19301

MR KRISTIAN J. HAMMOND
UNIV OF CHICAGO
COMPUTER SCIENCE DEPT/RY155
1100 E. 58TH STREET
CHICAGO IL 60637

MR. ROBERT FROST
MITRE CORP
WASHINGTON C3 CENTER, MS 644
7525 COLUMBIA ROAD
MCLEAN VA 22101-3481

RICK HAYES-ROTH
CIMPLEX-TEKNOWLEDGE
1810 EMBARCADERO RD
PALO ALTO CA 94303

RANDY GARRETT
INST FOR DEFENSE ANALYSES (IDA)
1301 N. BEAUREGARD STREET
ALEXANDRA VA 22311-1772

MR. JIM HENDLER
UNIV OF MARYLAND
DEPT OF COMPUTER SCIENCE
COLLEGE PARK MD 20742

MS. YOLANDA GIL
USC/ISI
4676 ADIRALTY WAY
MARINA DEL RAY CA 90292

DL-8
MR. JOHN LOWRENCE
SRI INTERNATIONAL
ARTIFICIAL INTELLIGENCE CENTER
333 RAvenSWOOD AVE
MENLO PARK CA 94025

DR. ALAN MEYROWITZ
NAVAL RESEARCH LABORATORY/CODE 5510
4555 OVERLOOK AVE
WASHINGTON DC 20375

ALICE MULVEHILL
MITRE CORPORATION
3 BURLINGTON RD
M/S K-302
BETFORD MA 01730

ROBERT MACGREGOR
USC/ISI
4676 ADMIRALTY WAY
MARINA DEL REY CA 90292

WILLIAM S. MARK, MGR AI CENTER
LOCKHEED MISSILES & SPACE CENTER
1801 PAGE MILL RD
PALO ALTO CA 94304-1211

RICHARD MARTIN
SOFTWARE ENGINEERING INSTITUTE
CARNEGIE MELLON UNIV
PITTSBURGH PA 16213

DREW MCDERMOTT
YALE COMPUTER SCIENCE DEPT
P.O. BOX 2158, YALE STATION
51 PROSPECT STREET
NEW HAVEN CT 06520

MS. CECILE PARIS
USC/ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292

DOUGLAS SMITH
KESTREL INSTITUTE
3260 HILLVIEW AVE
PALO ALTO CA 94304

DL-10
DR. AUSTIN TATE
AI APPLICATIONS INSTITUTE
UNIV OF EDINBURGH
80 SOUTH BRIDGE
EDINBURGH EH1 1HN - SCOTLAND

EDWARD THOMPSON
ARPA/SISTO
3701 N. FAIRFAX DR., 7TH FL
ARLINGTON VA 22209-1714

MR. STEPHEN F. SMITH
ROBOTICS INSTITUTE/CMU
SCHENLEY PRK
PITTSBURGH PA 15213

DR. ABRAHAM WAKSMAN
AFOSR/NM
110 DUNCAN AVE., SUITE B115
BOLLING AFB DC 20331-0001

JONATHAN P. STILLMAN
GENERAL ELECTRIC CRD
1 RIVER RD, RM K1-5C31A
P. O. BOX 8
SCHENECTADY NY 12345

MR. EDWARD C. T. WALKER
BBN SYSTEMS & TECHNOLOGIES
10 MOULTON STREET
CAMBRIDGE MA 02138

MR. BILL SWARTOUT
USC/ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292

GIO WIEDERHOLD
STANFORD UNIVERSITY
DEPT OF COMPUTER SCIENCE
439 MARGARET JACKS HALL
STANFORD CA 94305-2140

KATIA SYCARA/THE ROBOTICS INST
SCHOOL OF COMPUTER SCIENCE
CARNEGIE MELLON UNIV
DOHERTY HALL RM 3325
PITTSBURGH PA 15213

DL-11
<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR. DAVID E. WILKINS</td>
<td>SRI INTERNATIONAL ARTIFICIAL INTELLIGENCE CENTER 333 RAVENSWOOD AVE MENLO PARK CA 94025</td>
</tr>
<tr>
<td>DR. PATRICK WINSTON</td>
<td>MASS INSTITUTE OF TECHNOLOGY RM NE43-817 545 TECHNOLOGY SQUARE CAMBRIDGE MA 02139</td>
</tr>
<tr>
<td>HUA YANG</td>
<td>COMPUTER SCIENCE DEPT UNIV OF CALIFORNIA LOS ANGELES CA 90024</td>
</tr>
<tr>
<td>LTCOL DAVE NEYLAND</td>
<td>ARPA/ISTO 3701 N. FAIRFAX DRIVE, 7TH FLOOR ARLINGTON VA 22209-1714</td>
</tr>
<tr>
<td>MR. RICK SCHANTZ</td>
<td>BBN SYSTEMS &amp; TECHNOLOGIES 10 MOULTON STREET CAMBRIDGE MA 02138</td>
</tr>
<tr>
<td>LTC FRED M. RAWCLIFFE</td>
<td>USTRANSCOM/TCJS-SC 10 MOULTON STREET CAMBRIDGE MA 02138</td>
</tr>
<tr>
<td>ARPA/SISTO</td>
<td>ATTN: MR JOHN P. SCHILL 3701 N FAIRFAX DRIVE ARLINGTON VA 22203-1714</td>
</tr>
<tr>
<td>MR. DONALD F. ROBERTS</td>
<td>RL/C3CA 525 BROOKS ROAD GRIFFISS AFB NY 13441-4505</td>
</tr>
<tr>
<td>ALLEN SEARS</td>
<td>MITRE 7525 COLESHIRE DRIVE, STOP Z289 MCLEAN VA 22101</td>
</tr>
</tbody>
</table>

DL-12
DR CARLA LUOLOW  
ROME LABORATORY/C3CA  
525 BROOKS RD  
GRIFFISS AFB NY 13441-4505

DR MARK T. MAYBURY  
ASSOCIATE DIRECTOR OF AI CENTER  
ADVANCED INFO SYSTEMS TECH G041  
MITRE CORP, BURLINGTON RD, MS K-329  
BEDFORD MA 01730

MR DONALD P. MCKAY  
PARAMAX/UNISYS  
P.O. BOX 517  
PAOLI PA 19301

DR KAREN MYERS  
AI CENTER  
SRI INTERNATIONAL  
333 RAVENSWOOD  
MENLO PARK CA 94025

DR MARTHA E POLLACK  
DEPT OF COMPUTER SCIENCE  
UNIVERSITY OF PITTSBURGH  
PITTSBURGH PA 15260

RAJ REDDY  
SCHOOL OF COMPUTER SCIENCE  
CARNEGIE MELLON UNIVERSITY  
PITTSBURGH PA 15213

EDWINA RISSLAND  
DEPT OF COMPUTER & INFO SCIENCE  
UNIVERSITY OF MASSACHUSETTS  
AMHERST MA 01003

MR NORMAN SADEH  
CIMDS  
THE ROBOTICS INSTITUTE  
CARNEGIE MELLON UNIVERSITY  
PITTSBURGH PA 15213

MR ERIC TIFFANY  
ASCENT TECHNOLOGY INC.  
237 LONGVIEW TERRACE  
WILLIAMSTOWN MA 01267

DL-16
MANUELA VELOSO
CARNegie MELLON UNIVERSITY
SCHOOL OF COMPUTER SCIENCE
PITTSBURGH PA 15213-3391

MR DAN WELD
DEPT OF COMPUTER SCIENCE & ENG
MAIL STOP FR-35
UNIVERSITY OF WASHINGTON
SEATTLE WA 98195

MR CRAIG WIER
ARPA/SISTO
3701 N. FAIRFAX DR
ARLINGTON VA 22203

MR JOE ROBERTS
ISX CORPORATION
4301 N FAIRFAX DRIVE, SUITE 301
ARLINGTON VA 22203

COL JOHN A. WARDEN III
ASC/CC
225 CHENNAULT CIRCLE
MAXWELL AFB AL 36112-6426

DR TOM GARVEY
ARPA/SISTO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714

MR JOHN N. ENTZMINGER, JR.
ARPA/DIRO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714

LT COL ANTHONY WAISANEN, PHD
COMMAND ANALYSIS GROUP
HQ AIR MOBILITY COMMAND
402 SCOTT DRIVE, UNIT 3L3
SCOTT AFB IL 62225-5307

DIRECTOR
ARPA/SISTO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714

DL-17
Rome Laboratory

Customer Satisfaction Survey

Please complete this survey, and mail to RL/IMPS, 26 Electronic Pky, Griffiss AFB NY 13441-4514. Your assessment and feedback regarding this technical report will allow Rome Laboratory to have a vehicle to continuously improve our methods of research, publication, and customer satisfaction. Your assistance is greatly appreciated.

Thank You

Organization Name:_________________________(Optional)

Organization POC: ___________________________(Optional)

Address:_____________________________________

1. On a scale of 1 to 5 how would you rate the technology developed under this research?

5-Extremely Useful   1-Not Useful/Wasteful

Rating_____  

Please use the space below to comment on your rating. Please suggest improvements. Use the back of this sheet if necessary.

2. Do any specific areas of the report stand out as exceptional?

Yes____ No_____

If yes, please identify the area(s), and comment on what aspects make them "stand out."
3. Do any specific areas of the report stand out as inferior?

   Yes   No

   If yes, please identify the area(s), and comment on what aspects make them "stand out."

4. Please utilize the space below to comment on any other aspects of the report. Comments on both technical content and reporting format are desired.
MISSION
OF
ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.