AFRL-IF-RS-TR-2004-129
Final Technical Report
June 2004

DEVELOPMENT OF A FORMAL THEORY OF AGENT-BASED COMPUTING FOR SYSTEM EVALUATION AND SYSTEM-DESIGN GUIDANCE

University of Michigan

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. K550

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** June 2004	**3. REPORT TYPE AND DATES COVERED** **FINAL** **Aug 00 – Sep 03**

4. TITLE AND SUBTITLE DEVELOPMENT OF A FORMAL THEORY OF AGENT-BASED COMPUTING FOR SYSTEM EVALUATION AND SYSTEM-DESIGN GUIDANCE	**5. FUNDING NUMBERS** G - F30602-00-2-0621 PE - 62301E PR - TASK TA - 00 WU - 16
6. AUTHOR(S) Martha E. Pollack	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Michigan 3003 South State Street Ann Arbor MI 48109-1274	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency 3701 North Fairfax Drive Arlington VA 22203-1714 / AFRL/IFTB 525 Brooks Road Rome NY 13441-4505	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER** AFRLIF-RS-TR-2004-129

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Frank H. Born/IFTB/(315) 330-4726 Frank.Born@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT *APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.*	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(Maximum 200 Words)*

This report summarizes the research done on the DARPA-sponsored project on the Development of a Formal Theory of Agent-Based Computing for System Evaluation and System-Design Guidance, as part of the TASK program. The work was performed between September 2000 and September 2003, at the Artificial Intelligence Laboratory in the Department of Electrical Engineering and Computer Science at the University of Michigan. During the course of the project, significant advances have been made in the area of commitment strategies for autonomous agents, to enable such agents to manage sets of plans with rich temporal constraints in dynamic, uncertain environments. Specifically, we developed a set of computationally efficient techniques for both determining the consistency of sets of actions in order to decide whether or not newly introduced actions are compatible with existing commitments, and for merging new commitments into sets of existing ones. We also developed strategies for modifying a set of commitments in response to a new, incompatible action. Finally, we applied these computational techniques to various applications of interest to the TASK effort, including e-commerce, a “briefing agent”, and autonomous unmanned vehicles.

14. SUBJECT TERMS Software Agents, commitment strategy, Temporal Planning, plan management	**15. NUMBER OF PAGES** 124
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

1. Project Objectives

In the 21st century, the fundamental paradigm of computing will shift to one of intelligent, distributed agents, which are capable of efficiently processing asynchronous input from humans and from other computational agents, and of producing timely output. The intelligent distributed-agents paradigm will be especially valuable for a range of military applications that involve the processing of massive amounts of heterogeneous information, interrelated in complex ways, in time-critical situations. Such agent-based systems have four key characteristics. First, they receive inputs asynchronously, with additional input often arriving before the processing of previous inputs is complete. Second, the outputs of these systems may not be immediate, and so commitments to future events and reservations for future resources must be made. Third, there may be multiple ways of processing any input, where the decision among these depends on the commitments and reservations already made. Fourth, each system may be part of a larger "community" of systems, so that issues of coordination, competition, and communication come into play.

The goal of this project was to investigate the foundations of systems with these four characteristics, aiming towards the objective of developing a predictive theories of agent-system behavior that are useful both for designing agent systems and for predicting the performance of agent systems that are inherently ill-suited for full-fledged realistic testing (e.g, systems whose misperformance could have disastrous effects). In particular, in this project we focused on one critical aspect of agent-based systems: commitment strategies, which agent-based systems must employ to cope with asynchronous input, deferred output, and the significant demands on the computational processing that necessitate the careful management of computational resources. Commitments can be viewed as guarantees that certain actions—computational or external—will be performed by the individual agent, in a way that satisfies given execution constraints (These constraints may, for example, specify time or duration of performance, or quality of results produced).

To effectively manage a set of commitments, an agent must be able to perform the following computations:

- determine whether a new option for action (a new processing request) requires the abandonment or modification of any of its existing commitments;
- if it doesn't, determine the most efficient way to add a commitment to the new option;
- if it does, determine whether the set of prior commitments should be modified, and what the minimal-impact modification is.

Our emphasis on this project has been on the first two computations: we developed a set of computationally efficient techniques for both determining the consistency of sets of actions in order to determine whether or not newly introduced actions are compatible with existing commitments, and for merging new commitments into sets of existing ones. We also did work on the third topic, developing techniques for deciding whether and how to modify a set of commitments in response to a new, incompatible option in a way that minimizes a weighted function of the changes on the existing options and the cost of the resulting set of commitments. Additionally, we applied our algorithms to various applications that were of interest (at various times) to the TASK effort, including e-commerce, a “briefing agent”, and autonomous unmanned vehicles; we also leveraged this work by using the algorithms developed in the current project in the development of cognitive orthotic systems (which were supported by other projects).

2. Research Results

The principal results obtained during the project include the following (annotated with the publication in which they are reported in more detail):

- Formulation of the problem of commitment management as one of temporal constraint-satisfaction processing (CSP), and development of a general framework for doing consistency checking and update using temporal CSPs [5,6,10].

- Design, implementation, and experimental assessment of an efficient algorithm for checking the consistency of and updating tasks represented as disjunctive temporal problems, demonstrating speed-up of two orders of magnitude relative to the previous state-of-the-art [1,10].

- Development of an algorithm for flexible dispatch of plans encoded as DTPs that has a set of strong, provable properties [9,11,15].

- Formulation of a new temporal CSP representation, Conditional Temporal Problems (CTPs) used to represent tasks that include contingent outcomes, and development of algorithms for consistency-checking and update of tasks encoded as CTPs [2,15].

- Discovery of an undetected incompleteness in previous formalizations of the planning with contingencies, revealed by and corrected in the CTP formalism [2].

- Preliminary development of a strategy for dispatching plans encoded as simple temporal networks with uncertainty (STP-u’s), even when they are not consistent [4].

- Application of the consistency-checking and update algorithms to applications that were of interest to the TASK effort, including e-commerce, a “briefing agent”, and unmanned autonomous vehicles, as well as to cognitive orthotics systems [3,6,7,8,12].

- Development of an effective algorithm for replanning in response to a task that is not consistent with existing commitments, so as to maximize a weighted function of the changes on the existing options and the cost of the resulting set of commitments [13,14].

These main thrusts are very briefly described below.

2.1 Commitment Management as Temporal Constraint-Satisfaction Processing

The first task for a commitment-management component of a system is to determine whether or not a new option is potentially consistent with existing commitments, and if so, what additional constraints need to be made to ensure the consistency. The problem can be cast as one of temporal reasoning, in which the commitment set and the new option are modeled as sets of temporally grounded constraints on future actions, and the problem is to determine the consistency of the union of these sets. A side-effect of many consistency-checking algorithms is the inference of additional constraints that are needed to ensure consistency. A key question is when consistency-checking must be performed. In general, there are four triggers: the introduction of a new action; a modification to the existing commitments; the execution of an action; or the passing of a critical time boundary (e.g., the latest start time for one alternative way of performing an existing commitment). By modeling commitment management as a temporal constraint-satisfaction problem, we are able both to take advantage of known efficiencies in CSP processing, and to tailor the consistency-checking algorithm to the particular representational requirements of a given agent task-set (e.g., whether it requires the inclusion of disjunctive constraints; the modeling of causal uncertainty; and/or the modeling of temporal uncertainty).

See references [5,6,10].

2.2 Efficient Processing of Disjunctive Temporal Problems

For many years, consistency checking of sets of temporal constraints was limited to classes of constraints with restrictive expressiveness, particularly what are called Simple Temporal Problems (STPs). Although such problems can be solved with polynomial-time shortest-path algorithms, they include only atomic (non-disjunctive) constraints, and thus are not suitable for modeling advanced agent-based systems. Not only does the restriction to STPs preclude complex temporal constraints, but it also precludes least-commitment approaches to conflict resolution, since expressing a constraint of the form "Either do A before B or after C", i.e., promote or demote, is not expressible in an STP. To handle such constraints, one can employ a richer formalism: Disjunctive Temporal Problems (DTPs). Checking the consistency of a DTP is an NP-hard problem, and therefore heuristic techniques are pursued. We conducted a systematic analysis of previous approaches to DTP-solving, and used that as the basis of a new algorithm, called Epilitis, that is two orders of magnitude faster than previous DTP-solvers. Epilitis was fully implemented and has been made available to the research community.

See references [1,10].

2.3 Dispatch of Disjunctive Temporal Problems

By definition, agent systems need not only to manage their commitments, but also to execute them. The *dispatch problem* is the problem of deciding when to execute each action to which the agent has committed. When the commitments have temporal constraints, the dispatch problem may be non-trivial. As with consistency-checking, most prior work on dispatch focused on STPs. We developed a dispatch algorithm that is applicable to DTPs. We identified a set of properties that any dispatch algorithm must possess—it must be correct, deadlock-free, maximally flexible, and useful—and we proved that our algorithm has these properties.

See references [9,11,15].

2.4 Formulation of Conditional Temporal Problems

STPs and DTPs do not directly allow the encoding of actions that have uncertain outcomes. Yet often agents must deal with precisely such actions. To facilitate this, we developed a new formalism, called Conditional Temporal Problems (CTPs). CTPs are an extension of the standard temporal constraint-satisfaction problem, which (1) include observation nodes, and (2) attach labels to all nodes, indicating the situations in which the action they represent will be performed. The extended framework thus supports the modeling of conditional actions. We developed algorithms for consistency-checking with CTPs: importantly, consistency checking in a CTP can be achieved even while allowing for decisions about the precise timing of actions to be postponed until execution time, thereby adding flexibility and making it possible to dynamically adjust the commitments in response to observations made during execution. Our formulation of CTPs also led to a discovery of a previously undetected incompleteness in the conditional planning literature: it turns out that, even for plans without explicit quantitative temporal constraints, prior approaches will sometimes deem a planning problem unsolvable when in fact there is a solution to it. The use of the CTP formalism eliminates this problem.

See reference [2].

2.5 Dispatching Temporal Problems with Uncertainty

Where CTPs allow one to model causal uncertainty, an alternative formalism, STPU's (Simple Temporal Plans with Uncertainty) had been proposed in the literature to model temporal uncertainty, and an algorithm for assessing the consistency of STPU's had been developed. However, the algorithm provided no guidance to the agent in the case in which the STPU could not be guaranteed to succeed; yet, by definition, STPUs involve uncertain relations, and it may be very likely that a given STPU will be successfully executed even if this cannot be guaranteed. We thus addressed the question of what an agent should do with such an STPU, developing a set of three alternative approaches, one based on a binary search, one based on iterative tightening, and one based on an approximation of linear optimization. These approaches can be used both to provide lower bounds on the probability of successful execution, and to provide guidance to the agent about when to execute the actions involved.

See reference [4].

2.6 Applications

To demonstrate the viability of the representations and algorithms we developed, we applied them to work to various applications that were of interest to the TASK project, including E-commerce, a “briefing agent”, and analyses of clusters of simulated unmanned autonomous vehicles. We also leveraged the work done in the project by using the results obtained here in work we did *with the support of other contracts* to develop a cognitive orthotic system, which is a form of a plan-management agent used to assist people with memory decline in carrying out their activities effectively. Although this latter application is not within the scope of the current project, it is worth noting that we were able to build heavily on the techniques developed here in designing that agent.

See references [3,6,7,8,12].

2.7 Commitment Modification

When an agent determines that a new option is not consistent with its existing commitments, it needs to decide whether and when to modify those commitments to allow adoption of the new option. We developed an approach to commitment modification that builds on our previous work on cost-assessment in context. Specifically, it formalizes the notion of modification cost—or amount of change in a set of commitments—and provides a way for the agent to trade modification cost against overall cost in context of the revised plan. This work is still in progress, and we expect to publish our results within the next six months.

See references [13,14].

3. Publications Supported by the Project

Below are papers whose results were, at least in part, supported by the project.

1. Tsamardinos and M. E. Pollack, “Efficient Solution Techniques for Disjunctive Temporal Reasoning Problems,” to appear in *Artificial Intelligence,* 2003.

2. I. Tsamardinos, T. Vidal, and M. E. Pollack, “CTP: A New Constraint-Based Formalism for Conditional, Temporal Planning,” *Constraints: An International Journal*, 8(4): 365-383, 2003.

3. M. E. Pollack, L. Brown, D. Colbry, C. E. McCarthy, C. Orosz, B. Peintner, S. Ramakrishnan, and I. Tsamardinos, "Autominder: An Intelligent Cognitive Orthotic System for People with Memory Impairment," *Robotics and Autonomous Systems,* 44(3-4):273-282, 2003. *[Not supported by this project, but illustrates an application that leverages the work done in this project.]*

4. I. Tsamardinos, M. E. Pollack, and S. Ramakrishnan, "Assessing the Probability of Legal Execution of Plans with Temporal Uncertainty," *ICAPS Workshop on Planning under Uncertainty and Incomplete Information*, June 2003.

5. M. Beetz, M. Ghallab, J. Hertzberg, and M. E. Pollack, editors, *Plan-Based Control of Robotic Agents*, LNCS/LNAI #2466, Springer-Verlag, 2002.

6. M. E. Pollack, C. E. McCarthy, S. Ramakrishnan, and I. Tsamardinos, "Execution-Time Plan Management for a Cognitive Orthotic System," in M. Beetz et al., editors, *Plan-Based Control of Robotic Agents*, 2002.

7. M. E. Pollack, "Planning Technology for Intelligent Cognitive Orthotics," *6th International Conference on AI Planning and Scheduling*, Apr., 2002.

8. A. Berfield, P. Chrysanthis, I. Tsamardinos, M. E. Pollack, and S. Banerjee, "A Scheme for Integrating e-Services in Establishing Virtual Enterprises," *12th IEEE International Workshop on Research Issuess in Data Engineering,* Feb. 2002.

9. I. Tsamardinos, M. E. Pollack, and P. Ganchev, "Flexible Dispatch of Disjunctive Plans," in *6th European Conference on Planning,* Oct. 2001.

10. I. Tsamardinos, "*Constraint-Based Temporal Reasoning Algorithms, with Applications to Planning,"* University of Pittsburgh Intelligent Systems Program Ph.D. dissertation, Aug. 2001.

11. P. Ganchev, "*Flexibility Measures of Sets of Plans,*" University of Pittsburgh Intelligent Systems Program M.S. Project Report, June, 2001.

12. M. E. Pollack and J. F. Horty, "An Information Dynamics Research Exploration Framework: Briefing Agents," *DARPA/TASK Workshop*, Santa Fe, NM, April, 2001.

13. J. F. Horty and M. E. Pollack, "Evaluating New Options in the Context of Existing Plans," *Artificial Intelligence*, 127(2):199-220, 2001. *[Note: Much of the work on this paper was completed prior to start of the current contract.]*

14. J. Horty, and M. E. Pollack, "Minimal-Impact Replanning," in progress.

15. M. E. Pollack and I. Tsamardinos, "Dispatching Temporal Plans," in progress.

APPENDIX A

EFFICIENT SOLUTION TECHNIQUES FOR DISJUNCTIVE TEMPORAL REASONING PROBLEMS

Ioannis Tsamardinos
Department of Biomedical Informatics
Vanderbilt University
Ioannis.Tsamardinos@vanderbilt.edu

Martha E. Pollack
Computer Science and Engineering
University of Michigan
pollackm@eecs.umich.edu

Abstract

Over the past few years, a new constraint-based formalism for temporal reasoning has been developed to represent and reason about Disjunctive Temporal Problems (DTPs). The class of DTPs is significantly more expressive than other problems previously studied in constraint-based temporal reasoning. In this paper we present a new algorithm for DTP solving, called Epilitis, which integrates strategies for efficient DTP solving from the previous literature, including conflict-directed backjumping, removal of subsumed variables, and semantic branching, and further adds no-good recording as a central technique. We discuss the theoretical and technical issues that arise in successfully integrating this range of strategies with one another and with no-good recording in the context of DTP solving. Using an implementation of Epilitis, we explore the effectiveness of various combinations of strategies for solving DTPs, and based on this analysis we demonstrate that Epilitis can achieve a nearly two order-of-magnitude speed-up over the previously published algorithms on benchmark problems in the DTP literature.

1. Introduction

Expressive and efficient temporal reasoning is essential to a number of areas in Artificial Intelligence (AI). Over the past few years, a new constraint-based formalism for temporal reasoning has been developed to represent and reason about Disjunctive Temporal Problems (DTPs) [Stergiou and Koubarakis 1998; Armando, Castellini et al. 1999; Oddi and Cesta 2000]. The class of DTPs is significantly more expressive than other problems already studied in constraint-based temporal reasoning. It extends the well-known Simple Temporal Problem (STP) [Dechter, Meiri et al. 1991] by allowing disjunctions and the Temporal Constraint Satisfaction Problem (TCSP) *ibid.* by removing restrictions on the form of allowable disjunctions.

Formally, a Disjunctive Temporal Problem (DTP) is a pair <*V*, *C*> where *V* is a set of temporal variables and *C* is a set of constraints among the variables. Every constraint C_i . C is of the form:

$$c_{i1} \;.\; \dots \;.\; c_{in}$$

where in turn, each c_{ij} is of the form $x - y = b$; x,y . V and b . P.[1] Constraint c_{ij} is called the *j*th *disjunct* of the *i*th constraint. A solution to a DTP is an assignment to each variable in V such that all the constraints in C are satisfied. If a DTP has at least one solution, it is *consistent*. Notice that each constraint C_i may involve more than two temporal variables, in which case it is not a binary constraint. Only DTPs, and not STPs or TCSPs, allow difference constraints of arbitrary arity.

The increased expressivity of DTPs makes them a suitable model for many planning and scheduling problems. Plan generation, plan merging, job-shop scheduling, and even temporal reasoning under certain forms of uncertainty can all be modeled as DTPs. As a motivating example, consider a temporal plan with steps A and B that both represent actions requiring the same unary resource, e.g., they both use the same printer. In this case, the executions of A and B should not overlap. We can encode this fact as a DTP constraint by defining the DTP variables *start(A)*, *start(B), end(A),* and *end(B)* associated with the instants of starting and ending A and B. The DTP constraint then is the following:

$$end(A) - start(B) = 0 \;.\; end(B) - start(A) = 0$$

i.e. either A finishes before B starts or vice versa. It is easy to see how such constraints can encode classical threat resolution in planning. Analysis of the DTP defining this plan can reveal whether it is feasible: the DTP is consistent if and only if there is a way to execute the plan such that the deadlines are all met, and the unary resource (the printer) is never used more than once at the same time.

Threat-resolution constraints in planning, as just described, can also be encoded as binary constraints between intervals. There are situations however when this is not possible and higher arity constraints are required and thus cannot be expressed (in the general case) by any other current formalism but the DTP. For example consider reasoning about the following scenario: "if you can, stop by the post-office for 10-15 minutes, then take route A for 10-15 minutes, or else take route B for 10-15 minutes." If we use *PO* to denote the fluent 'in the post-office', then this scenario can be represented as the following constraints:

$$(10 = end(PO) - start(PO) = 15 \;.\; 10 = end(A) - start(B) = 15 \;.\; end(PO) = start(A)) \;.$$
$$10 = end(B) - start(A) = 15$$

In turn, this can be directly converted to DTP form.

The principal approach to DTP solving taken in the literature has been to convert the original problem to one of selecting one disjunct, $x_i - x_j = b_{ji}$ from each constraint C_i . C and then checking that the set of selected disjuncts forms a consistent STP. Checking the consistency of and finding a solution to an STP can be performed in polynomial time using shortest-path algorithms [Dechter, Meiri et al. 1991]. The computational complexity in DTP solving derives from fact that there are exponentially many sets of selected disjuncts that may need to be considered; the challenge is to find ways to efficiently explore the space of disjunct combinations. This has been done by casting the disjunct selection problem as a constraint satisfaction processing (CSP) problem [Stergiou and Koubarakis 2000] [Oddi and Cesta 2000] or a satisfiability (SAT) problem [Armando, Castellini et al. 1999].

[1] As is standard in the literature, in this paper we will make the assumption, without loss of generality, that the bounding values *b* are integers.

In this paper, we present a new algorithm and heuristics for DTP solving, embodied in a system we call ***Epilitis***[2], which integrates strategies for efficient DTP solving from the previous literature, including conflict-directed backjumping, removal of subsumed variables, and semantic branching, and further adds no-good recording [Schiex and Verfaillie 1994] as a central technique. We discuss the theoretical and technical issues that arise in successfully integrating the previous strategies with one another and with no-good recording. Using an implementation of Epilitis, we explore the effectiveness of various combinations of strategies for solving DTPs, and based on this analysis, we demonstrate that Epilitis achieves a nearly two-order-of-magnitude speed-up over the previously published algorithms on benchmark problems from the DTP literature. This result is based on speed comparisons because we demonstrate that counting the number of forward-checks (also called consistency-checks), as is commonly done in the literature, is not an accurately descriptive measure of performance.

The overall structure of the paper is as follows. In Section 2 we present necessary background information on the DTP and DTP solving. Section 3 explains all previous methods for pruning the search for a DTP solution. Section 4 presents necessary background information on no-good recording. Section 5 uses the background material presented to describe the Epilitis system and the underlying algorithms we used. In Section 6 we present our experiments with Epilitis. Section 7 reviews related work in the field. Section 8 concludes the paper with an overall discussion and presentation of future work.

2. Solving Disjunctive Temporal Problems

2.1 The Basic Approach

We begin by reviewing a simpler class of temporal problems: the Simple Temporal Problems (STPs). An STP, like a DTP, is a pair $<V, C>$, where V is again a set of temporal variables; for an STP, however, C is a single constraint of the form $x - y = b$, where x,y . V. Because an STP contains only binary constraints, it can be represented with a weighted graph called a Simple Temporal Network (STN), in which an edge (y, x) with weight b_{yx} exists between two nodes *iff* there is a constraint $\{x - y = b_{yx}\}$. C . Polynomial-time algorithms can be used to compute the all-pairs shortest path matrix, or *distance array* of the STN. We denote the distance (shortest path) between two nodes x and y as d_{xy}. The concept of the distance is important because in a consistent STP, d_{xy} is the largest number for which the constraint $y - x = d_{xy}$ holds in every solution. In addition, an STP is consistent if and only if for every node x in its associated STN, $d_{xx} = 0$, which means that there are no negative cycles [Dechter, Meiri et al. 1991].

A DTP can be viewed as encoding a collection of alternative STPs. To see this, recall that each constraint in a DTP is a disjunction of (one or more) STP-style inequalities. Let c_{ij} be the j^{th} disjunct of the i^{th} constraint of the DTP. If we select one disjunct c_{ij} from each constraint C_i, then the set of selected disjuncts forms an STP, which we will call a *component STP* of the given DTP. It is easy see that a DTP D is consistent if and only if it contains at least one consistent

[2] From the Greek word *Επιλυτής* (solver).

component STP. Moreover, any solution to a consistent component STP of D is also a solution to D itself. Because only polynomial time is required both to check the consistency of an STP, and, if consistent, extract a solution of it, in the remainder of this paper we will say that *the solution of a given DTP is any consistent component STP of it.* When we need to refer to an actual assignment of numbers to the time points in the DTP, we will call this an *exact solution.* A consistent component STP represents a number of exact DTP solutions. This is particularly important in planning since it provides execution flexibility. The consistent component STP can then be executed as described in [Tsamardinos 1998].

Definition 1: A time assignment to the time-points of a DTP is called ***an exact solution*** of the DTP. A consistent component of a DTP is called a ***solution*** of the DTP.

	Original CSP (the DTP)	Meta-CSP
"Variables"	x, y, z … (Time Points)	One variable C_i for each constraint C_i of the original DTP (Variables)
Domains	(-8 , +8) for all variables	$D(C_i)=\{c_{i1}, \ldots, c_{im}\}$ (Sets of Constraints)
"Constraints"	$x_1 - y_1 = b_1$. …. $x_n - y_n = b_n$ i.e c_{i1} . …. c_{in} (Disjunctions of Constraints)	Implicitly defined by the underlying semantics of the values in each domain.

Table 1: Correspondence between the DTP and the meta-CSP

All existing algorithms for DTP solving, including the one we present in this paper, work by searching for a consistent component STP S from a given DTP D rather than attempting to search directly for a consistent assignment to the nodes of D. The process of finding S can itself be modeled as one of constraint satisfaction processing. Because the original DTP is itself also a CSP problem, we will refer to the problem of extracting a consistent component STP as the *meta-CSP problem.* The meta-CSP contains one variable for each constraint C_i in the DTP. The domain of C_i is the set of disjuncts in the original DTP constraint C_i. The constraints in the meta-CSP are not given explicitly, but must be inferred: an assignment satisfies the meta-CSP constraints *iff* the assignment corresponds to a component STP that is consistent. For instance, if the variable C_i is assigned the value $x - y = 5$ it would be inconsistent to extend that assignment so that some other variable C_j is assigned the value $y - x = -6$.

In this paper, we will refer to the variables of the DTP as ***time points*** and will reserve the term ***variables*** for the meta-CSP. We will use the terms ***constraint*** and ***value*** interchangeably, to refer to a single, non-disjunctive constraint c_{ij}: such constraints (values) constitute the domains of the meta-CSP variables. Finally, we will reserve term ***node*** to refer to the nodes of a CSP tree search, which we will typically be performing for the meta-CSP—recall that we do not perform direct CSP processing on the DTP. The relationship between the original CSP (the DTP) and the meta-CSP (which aims to find a consistent component STP) is summarized in Table 1.

A typical forward-checking CSP algorithm, shown in Figure 1 can be used to solve a DTP—or more precisely, to solve its meta-CSP. The algorithm takes two parameters: A, de-

```
Basic-DTP(A, U)
1.    If U=¬ stop and report A as a solution.
2.    C .  select-variable(U), U'.  U-{C}
3.    For each value c of d(C) in some order
4.        A'=A. {C.  c}
5.        If forward-check(A', U')
6.            Basic-DTP(A', U')
7.        EndIf
8.        un-forward(U')
9.    EndFor
10.   Return failure

forward-check(A, U)
11.   For each variable C in U
12.       For each value c  in d(C)
13.       If not STP-consistency-check(A .  {C.  c})
14.           Remove c from d(C)
15.           If d(C) = ¬
16.               return false
17.           EndIf
18.       EndIf
19.   EndFor
20.   Return true
```

Figure 1: The Basic DTP algorithm

noting the set of already assigned variables and their assigned values, and U, the set of as-yet unassigned variables. The initial call to solve a DTP $<V, C>$ should be made with $A = ¬$ and $U = C$, and the initial ***current domains*** $d(C)$ should be initialized to the ***original domains*** $D(C_i)$, i.e. to the set of constraints that constitute the disjuncts in C_i in the DTP (see Table 1).

The function **select-variable** heuristically selects the next variable to which to make an assignment; the decision about how to make that selection is left unspecified in the generic algorithm, but we discuss it further in Section 6.4. The function **forward-check**(A, U) performs forward checking, i.e., it removes from the domains of the variables still in U all those values that are inconsistent with the current assignment A, returning *false* if, as a result, one or more variables in U has a domain reduced to ¬. Note that in DTP-solving, **forward-check**[3] operates by checking the consistency of an STP (specifically, a component STP of the DTP), which, as mentioned above, requires only polynomial time. If forward-checking fails, then the function **un-forward** restores the domains of the variables to those before the last call to **forward-check**.

2.2 Improved Forward-Checking

A large portion of the computation in algorithm **Basic-DTP** is spent at line 13 in **forward-check,** where each value of each variable is added to the set of constraints of the current as-

[3] We only describe DTP solvers using forward-check because it has been proven very efficient in DTP literature and it is a standard component of every DTP solver. In addition, not using forward-checking dramatically reduced efficiency in preliminary experiments in our lab.

```
forward-check-with-FC(S, U)
12.   For each variable C in U
13.       For each value $c : x - y = b_{xv}$ in d(C)
14.       If $b_{xy}$ + distance (y, x, S)
15.           Remove c from d(C)
16.           If d(C) = ¬
17.               return false
18.           EndIf
19.       EndIf
20.   EndFor
21.   Return true
```

Figure 2: Forward checking in STPs using the FC condition

signment A and checked for STP-consistency. STP-consistency checking takes time $O(|V|^3)$ where $|V|$ is the number of time-points; thus forward-check takes time $O(v|V|^3)$ on each node, where v is the number of values to forward check. Fortunately, there is a computationally less costly way of achieving forward checking of values, based on the following theorem:[4]

Theorem 1: A value $c_{ij} : y - x = b_{xy}$ is inconsistent with a consistent STP S (that is, S . c_{ij} is inconsistent) if and only if the following condition holds:

$$b_{xy} + d_{yx}(S) < 0 \quad (\textbf{FC-condition})$$

where $d_{yx}(S)$ is the distance between nodes y and x in S.

Theorem 1 (the proof is in Appendix A) indicates that to forward-check a particular value $y - x = b_{xy}$ against an assignment A, we just need to check the FC-condition. In turn, this requires calculating the distances d_{yx} in STP S for all nodes x and y. One method for calculating all these distances efficiently is to calculate the distance array; this is equivalent to running full path consistency, which has time complexity time $O(|V|^3)$. Once the distance array has been calculated, the distance between any two nodes y and x can be recovered by matrix lookup in constant time; hence the overall time required for each node is $O(|V|^3 + v)$, where v is the number of values to be forward checked. An alternative technique is to compute directional path consistency [Chleq 1995] where only part of the shortest path array is cached, in a manner that permits the uncached distances to be recovered in time at most $O(|V|)$.

To modify the main algorithm in Figure 1 with improved forward checking, we only need do two things. First, we replace the forward-checking routine with one that uses the FC-condition, as shown in Figure 2. Second, we add one line to the main program (Basic-DTP); specifically,

S'=**maintain-consistency***(c, S)*

should be inserted between lines 4 and 5. Each time a new variable is assigned a value $\{C. \quad c\}$, the constraint is propagated in S by **maintain-consistency**(*c, S*), which can be implemented with either full path consistency or with directional path consistency, as described above.

[4] This theorem was suggested, but not proved, in [Oddi and Cesta 2000] see Appendix A for its proof.

The comparison of overall complexity in each node does not yield an obvious "best" approach. As already noted, (i) the basic **forward-check** procedure requires $O(v|V|^3)$ time for each node in the CSP search tree; (ii) computing full path consistency and checking the FC-condition requires $O(|V|^3 + v)$; and (iii) computing directional path consistency for FC-checking requires $O(|V|^3 + v|V|)$. In addition, since assignment A is built incrementally by adding constraints on each new node, we can use incremental versions of these previous techniques to build S, namely *incremental full path consistency* **(IFPC)** [Mohr and Henderson 1986] and *incremental directional path consistency* **(IDPC)** [Chleq 1995]**.** The incremental versions drop the exponent in all the above complexities to quadratic and so (i) takes time $O(v|V|^2)$, (ii) takes time $O(|V|^2 + v)$ and (iii) $O(|V|^2 + v|V|)$. Thus, the worst-case comparison favors maintaining full path consistency (i.e. the distance array)[5]. The average case comparison cannot easily be resolved theoretically and further experiments are required to determine under which conditions each method is the best. In our experiments, reported below in Section 6 we used method (ii), maintaining the distance array at every node.

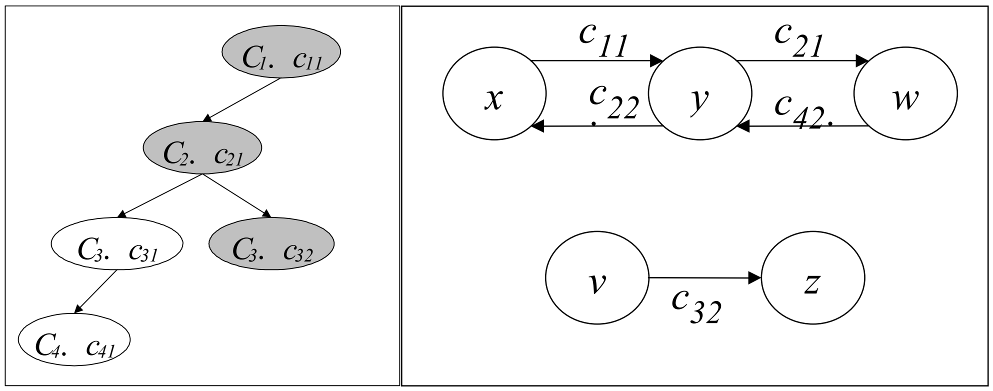
3. Previous Pruning Techniques for DTP Solving

Once the DTP problem has been cast as one of solving a meta-CSP, a number of different backtracking search techniques can be used to increase efficiency by early pruning of dead-end branches. The idea in pruning techniques is to utilize the underlying semantics of the values of the meta-CSP, namely the fact that they express constraints on some STP, to make inferences regarding the infeasibility of certain regions of the search. In this section we describe three methods previously considered in the literature: ***Conflict-Directed Backjumping*** **(CDB)** (used in [Stergiou and Koubarakis 2000] ***Semantic Branching*** **(SB)** (used in [Oddi and Cesta 2000] and [Armando, Castellini et al. 1999]), and ***Removal of Subsumed Variables*** **(RSV)** (used in [Armando, Castellini et al. 1999]). In the next section we will add ***No-good Recording*** **(NR)** (also called no-good learning), and hence throughout both these sections we will be particularly concerned with the theoretical and technical issues that arise in successfully integrating these pruning strategies with one another and with no-good recording.

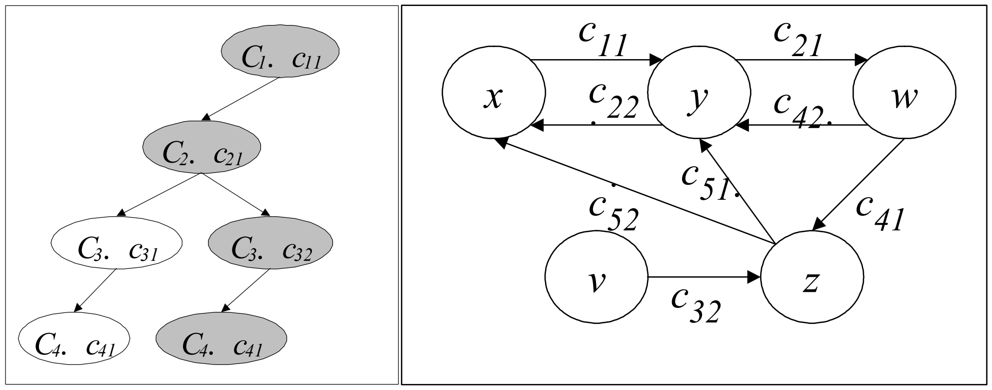
3.1 Conflict-Directed Backjumping for DTPs

The simplest algorithms for solving CSP problems rely on chronological backtracking, in which the failure of a partial assignment of values to variables results in backtracking to the point in the search just before the most recent assignment of a value to a variable was made. Previous work has shown that backtracking can be made more efficient by instead restarting the search at a more carefully selected point: techniques developed for this include Dynamic Backtracking [Ginsberg 1993], and Conflict Directed Backjumping (***CDB***) [Prosser 1993; Chen and Beek 2001]. In these approaches, when a dead end is encountered, the search backtracks to the most recently assigned variable that is *related* to the failure. The variables that are unrelated to the failure are backjumped over, since trying to assign different values for them will result in the same dead end.

[5] This is because the worst-case bounds are tight.



(a)



(b)

Figure 3: The chronological backtracking algorithm on a DTP.

It is obvious that to implement CDB, it is necessary to be able to identify the culprit of the failures, i.e., the variables that participate in the constraints that lead to failure. Stergiou and Koubarakis [Stergiou and Koubarakis 2000] present a method for culprit identification in DTP solving, which they call the *dependency pointers scheme*. This scheme is based on the fact that, during DTP solving, whenever an assignment *A* is extended to *A'*=*A*. {*C*. *c*}, forward checking is performed. If a value *c'* is removed from some domain, then the most recent value assignment, {*C*. *c*}, must directly contribute to its removal. In the approach of Stergiou and Koubarakis, a dependency pointer from *c'* to *c* is stored. If the algorithm subsequently needs to backtrack because the domain of some variable has been reduced to ¬, the algorithm checks the dependency pointers for values that were removed from that variable's domain, and follows the one that points to the most recently assigned variable, thereby backjumping over any irrelevant variables.

Although the dependency pointer scheme achieves CDB, it does not integrate well with semantic branching and no-good recording. We thus developed an alternative scheme for calculating the culprit of a failure. Our technique returns the variables of the current assignment that are involved in the failure; these can be used in a manner similar to dependency pointers, to backjump to the most recent relevant variable. Additionally, however, the returned set of variables can be used as justifications in no-good recording, as described in Section 4.

```
justification-value($c : y - x = b$ , $S$)
1. $p$ = shotest-path( $y$, $x$, $S$)
2. Return vars($p$ .   $c$}
```

Figure 4: Function justification-value

Our approach is straightforward, and builds directly on the fact that backtracking is required only when **forward-check** has reduced the domain of some variable to the empty set by removing every value c_j of that domain. This in turn implies that every c_i that was in the domain is part of a negative cycle p_i formed by constraints c_1 , …, c_k. We introduce the technique with an example.

Example 1: Figure 3 illustrates the processing of the following DTP:

$C_1 : \{c_{11} : y - x = 5\}$
$C_2 : \{c_{21} : w - y = 5\}$. $\{c_{22} : x - y = -10\}$. $\{c_{23} : z - y = 5\}$
$C_3 : \{c_{31} : v - x = 5\}$. $\{c_{32} : z - v = 10\}$
$C_4 : \{c_{41} : z - w = 5\}$. $\{c_{42} : y - w = -10\}$
$C_5 : \{c_{51} : y - z = -20\}$. $\{c_{52} : x - z = -20\}$

In the figure, the top two boxes (a) represent a "snapshot" of the DTP solving process: the left-hand side shows the meta-CSP search tree, and the right-hand side shows the STP entailed by the current assignment. The bottom two boxes (b) show a snapshot later in the process. At the time of Figure 3(a), assignments have been made to C_1, C_2, and C_3. The assignments chosen are indicated by the gray ovals while the white ovals indicate already explored nodes. Note that values that have been ruled out by forward-checking are crossed out in the STP diagrams. For instance, the assignment of c_{11} ($y - x = 5$) to C_1 rules out the possibility of assigning c_{22} ($x - y = -10$) to C_2, and so this value is crossed out in the right-hand part of Figure 3(a).

Figure 3(b) shows a later point in the processing, by which an assignment has also been made to C_4 (specifically C_4 . c_{41}). At this point, forward-checking will eliminate both possible values for C_5, because they participate in negative cycles. These cycles are independent of the assignment made to C_3, which should thus be backjumped over. That is, c_{51} and c_{52} are removed from d(C$_5$) because they form the negative cycles (in the STP): $p_1 = (c_{21}, c_{41}, c_{51})$ and $p_2 = (c_{11}, c_{21}, c_{41}, c_{52})$. The variables that participate in the failure then are **vars**(p_1) .   **vars**(p_2) = $\{C_2, C_4, C_5\}$. $\{C_1, C_2, C_4, C_5\} = \{C_1, C_2, C_4, C_5\}$, where **vars**($p$) are the variables whose value assignments are the constraints in p.

It is apparent that our technique requires the identification of a negative cycle for each removed value by forward check. This can be implemented by maintaining a predecessor array[6] [Cormen, Leiserson et al. 1990] when calculating the shortest path array. Entry $<i,j>$ of the predecessor array contains nil when $i=j$; otherwise it is a predecessor of j on the shortest path from i. It should be updated by the function **maintain-consistency**, which can be done without changing the time complexity of the function. When a value $c : y - x = b$ completes a negative cycle (i.e. the FC-condition holds), we follow the predecessor array to retrieve the shortest path p from y to x and return **vars**(p .   (x, y)), where (x, y) is the edge from x to y. The pseudo-

[6] Recall that the predecessor array stores a predecessor of j on the shortest path from i in all entries $<i,j>$ that are not on the main diagonal.

$C_1 : \{c_{11} : y - x = 5\}.\ \ \{c_{12} : w - y = -10\}$
$C_2 : \{c_{21} : x - z = 5\}$
$C_3 : \{c_{31} : y - z = 15\}\ .\ \ \{c_{32} : z - v = 10\}$
$C_4 : \{c_{41} : z - v = 5\}\ .\ \ \{c_{42} : y - w = -10\}$
$C_5 : \{c_{51} : v - y = -20\}\ .\ \ \{c_{52} : z - x = -10\}$
$C_6 : \{c_{61} : z - v = 2\}\ .\ \ \{c_{62} : x - y = -10\}$

Figure 5: Example DTP for removal of subsumed variables and semantic branching.

code for implementing this approach is given in Figure 4: the function **justification-value** returns the justification (i.e. the culprit set of variables) for the removal of value $c : y - x = b$ from the domain of its variable, given an STP S that corresponds to the current assignment.

3.2 Removal of Subsumed Variables

The main idea of the heuristic that we will call ***Removal of Subsumed Variables (RSV)*** is that if a disjunct c_{ij} of a variable C_i is already satisfied by the current assignment A, there is no reason to try other values in the variable's domain under assignment A because either (i) the current assignment leads to a solution, and since c_{ij} is already satisfied under A, C_i is satisfied in the solution, or (ii) there are no solutions under A and trying other values for C_i will only restrict A even further, with no possibility of discovering a solution. We now proceed by formalizing this idea[7].

Definition 2: ***A value c_{ij} is subsumed by an STP*** S (equivalently by an assignment A that implies S) if and only if the constraint c_{ij} always holds in any exact solution of S. (Recall that an exact solution to a DTP D is an assignment of numbers to the time points in D.) A ***variable C_i is subsumed by an STP*** S if and only if there is a value c_{ij} in the domain of C_i that is subsumed by S.

Theorem 2: A value $c_{ij} : y - x = b$ is subsumed by an STP S if and only if $d_{xy}(S) = b$ (***Subsumption-Condition***), where $d_{xy}(S)$ is the distance between x and y in S.

Theorem 3: Let $D=<V, C>$ be a DTP, let A be an assignment on D (i.e. a component STP), and let C_i be a variable subsumed by A. Then A is a solution of D if and only if it is a solution of $D'=<V, C - C_i>$.

Corollary 1: Let A be a partial assignment during a DTP search, U be the unassigned variables, and Sub be the set of subsumed variables in U. If A can be extended to a solution over variables in $U - Sub$, it can be extended to a solution over variables in U. In other words, we can remove the subsumed variables from the unassigned variables during search. The solution to the reduced problem is a solution to the original one.

The proofs for the above theorems and corollaries are in Appendix A.

[7] RSV was first used by [Oddi and Cesta 2000] but without providing a proof.

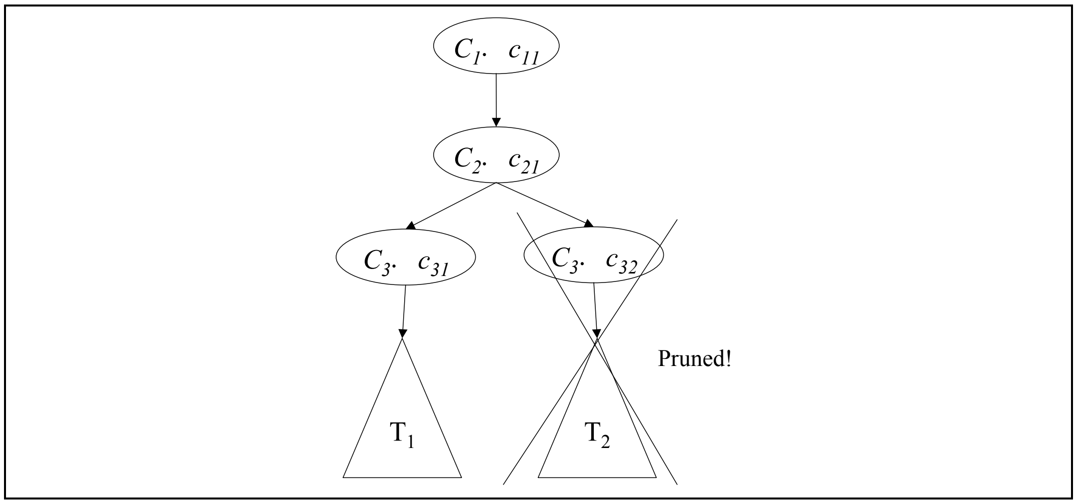


Figure 6: The search tree showing the effects of the removal of subsumed variables.

Example 2: The ramifications of the above corollary are shown pictorially Figure 6, which depicts a search *without* the use of RSV for a solution to the DTP in Figure 5. The variables are considered in order (1-6). Initially, $A_1 = \{C_1 .\ \ c_{11}\}$. This is then extended to $A_2 = \{C_1 .\ \ c_{11} , C_2 .\ \ c_{21}\}$. Without removing the subsumed variables, the next assignment would be $A_3 = \{C_1 .\ \ c_{11} , C_2 .\ \ c_{21} , C_3 .\ \ c_{31}\}$. Notice though that value c_{31} , and thus variable C_3 is subsumed by A_2, because together c_{11}*:* $y - x = 5$ and c_{21} : $x - z = 5$ imply that y-x=10, which subsumes the constraint c_{31} : $y - z = 15$. Thus, by Corollary 1 C_3 can safely be removed from the search underneath the subtree of A_2. Suppose, however, that it is not removed. Then the search will proceed as in Figure 6. When the search of subtree T_1 in figure fails, as it will in this particular example, the search continues by trying the other value of C_3 and so $A_4 = \{C_1 .\ \ c_{11} , C_2 .\ \ c_{21} , C_3 .\ \ c_{32}\}$. By Corollary 1, since A_3 has failed, A_4 will fail too. By removing the subsumed variable C_3 in this particular example, subtree T_2 and the node corresponding to A_4 in the figure would have been safely pruned.[8]

3.3 *Semantic Branching*

A third pruning method used in solving DTPs is semantic branching (***SB***), which has been shown to be very effective [Armando, Castellini et al. 1999]. Like *RSV*, *SB* relies on the semantics of the constraints in the DTP, i.e., on the fact that they encode numeric inequalities. The basic idea of semantic branching is the following. Suppose that during search the assignment A . $\{C_i .\ \ c_{ij}\}$ is expanded in every possible way but it leads to no solution. That means that in any solution that is an extension of A, if there is any, the constraint c_{ij} does not hold. Thus, the negation of this constraint has to hold in any such solution. In other words, if c_{ij} is the constraint $x - y = b$ and we know c_{ij} does not hold, then in any solution that is an extension of assignment A, $\leftarrow c_{ij}$ has to be true, i.e. it must be the case that $y - x < -b$. Thus, when

[8] The node corresponding to A_3 would also have been pruned.

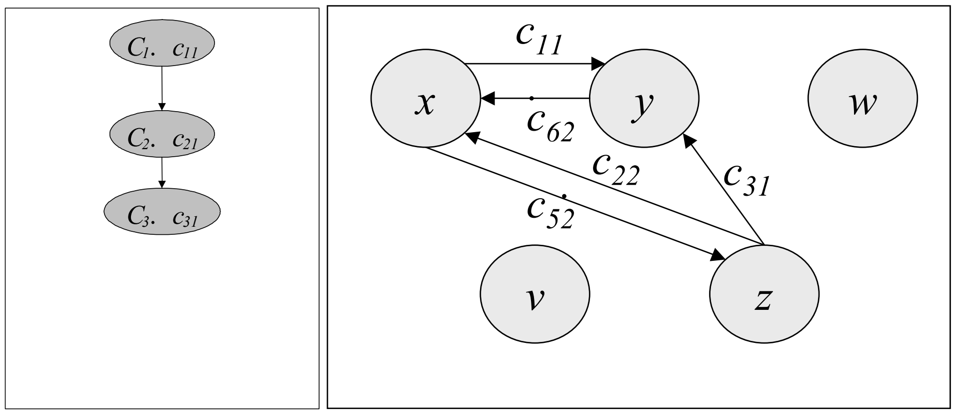


Figure 8: Semantic Branching example (a)

search "branches" after failing to extend A . $\{C_i$. $c_{ij}\}$ to a solution, and tries a different value for C_i for the rest of the search under A, we can assume ←c_{ij} holds.[9] The constraint ←c_{ij} often tightens the STP that corresponds to the current assignment A; explicitly adding this constraint can lead to values in other variable domains being removed earlier than they otherwise would have been.

Notice that with SB the current assignment A at any point in processing no longer stands in a one-to-one correspondence with an STP S. Instead S is the union of the values assigned to variables in A *and* all the current semantic branching constraints.

Example 3: To see how SB prunes the search space, we compare the search space for the DTP of Figure 5 without and then with semantic branching. Suppose the algorithm has already assigned $A_1 = \{C_1$. c_{11}, C_2 . c_{21}, C_3 . $c_{31}\}$ as shown in Figure 8. (On the left is the search of the meta-CSP and on the right is the implied STP.) The x-

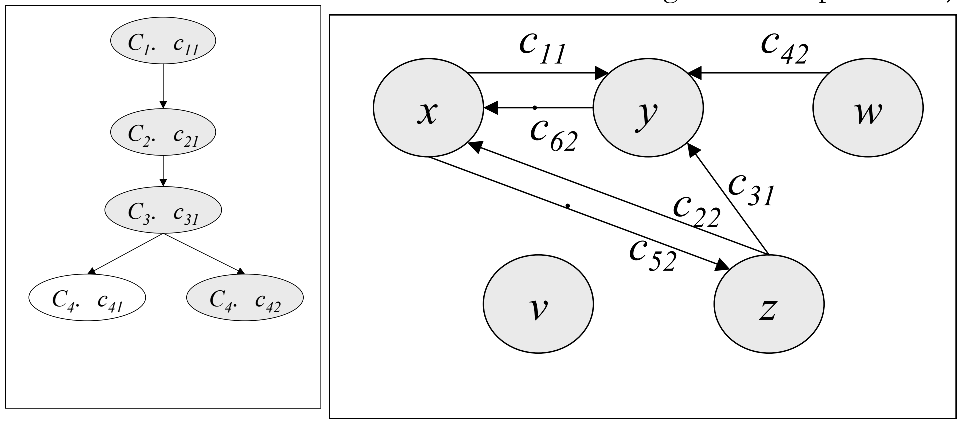


Figure 7: Semantic Branching Example (c)

crossed edges are the ones removed by forward checking while the filled nodes are the ones that belong to the current assignment. In Figure 9 the assignment is extended to $A_2 = \{C_1$. c_{11}, C_2 . c_{21}, C_3 . c_{31}, C_4 . $c_{41}\}$. This assignment fails because both values in $D(C_5)$ are removed.

[9] The implementation of semantic branching is discussed in Sec. 6.6.

The search then continues by trying a different value for C_4 (Figure 7). Finally, the search reaches a dead end again because both values in $D(C_6)$ are removed (Figure 10), after which it will backtracks back to node C_3 and continue the search.

Had we used semantic branching however, when we branched to try the second value of C4 we would have explicitly added the constraint ←c41, as shown in boldface in Figure 11. The constraint ←c41 allows forward checking to eliminate value c61 immediately, thus reaching a dead end. In this simple example, SB prunes only one node, the one that assigns C5 .   c51 (last node in left picture of Figure 10), but in general SB can prune an arbitrarily large number of nodes.

As is noted in [Oddi and Cesta 2000], Semantic Branching is only useful when the disjuncts in each constraint are not mutually exclusive. For example, in scheduling applications where

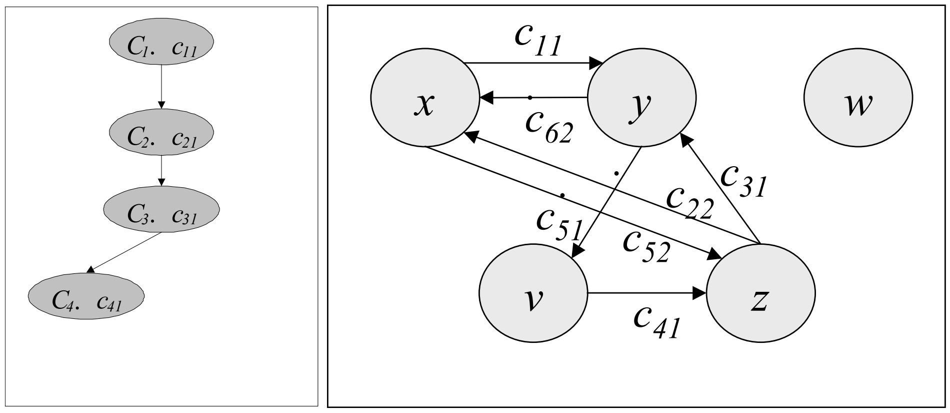


Figure 9: Semantic Branching example (b)

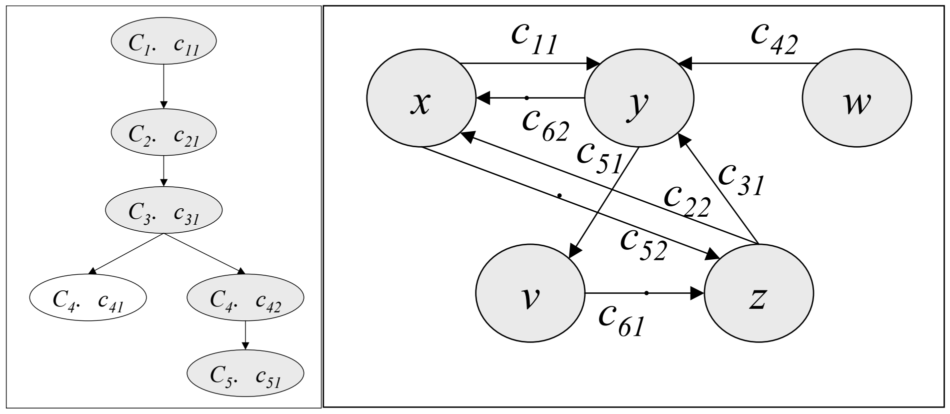


Figure 10: Semantic Branching example (d)

the constraints are typically of the form {$A < B$ or $B < A$}, when the first disjunct fails, SB will add its negation A > B, having no pruning effect, since the next disjunct $B < A$ is the same constraint.

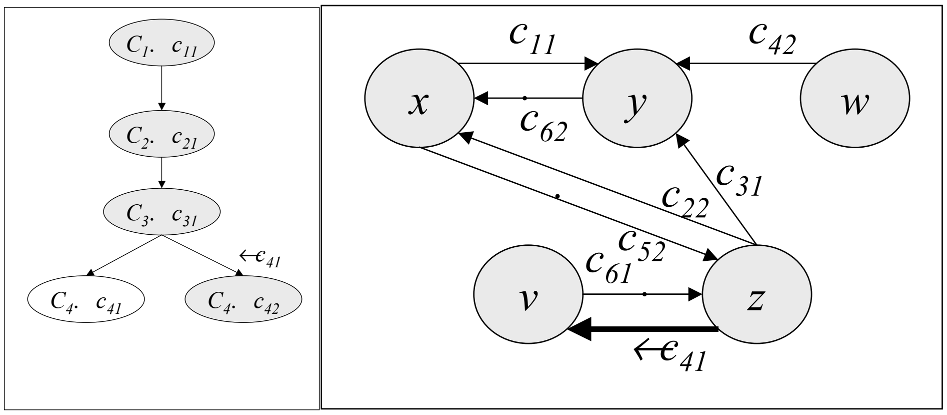


Figure 11: Semantic Branching example (e)

4. No-good Recording

No-good recording (also called no-good learning) is a powerful pruning technique for solving general CSPs [Dechter 1990; Frost and Dechter 1994; Ginsberg and McAllester 1994; Schiex and Verfaillie 1994; Schiex and Verfaillie 1994; Yokoo 1994; Dechter and Frost 1999] and SAT problems [Roberto J. Bayardo and Schrag 1977]. In this section, we adapt this technique to DTP solving. Intuitively, a no-good is an assignment of the variables that cannot lead to a solution, and is thus either an induced or explicit constraint of the CSP. It is important not to confuse no-goods with semantic branching constraints. No-goods are constraints of the meta-CSP, while SB constraints are constraints of the component STP associated with one particular (possibly partial) assignment to the variables of the meta-CSP.

In our Epilitis algorithm we use no-goods for two purposes: (i) for pruning the search space and (ii) as heuristic information to estimate which variables constrain the remaining search space the most. This section deals with the former, while Section 6.4 with the later.

We begin by defining no-goods in general, for an arbitrary CSP <*V, C*>. In our definitions, we will use C_X . *C* to denote the constraints in *C* that involve only the variables in *X*, where *X* . *V*.

Definition 3: A ***no-good*** of CSP <*V, C*> is a pair <*A, J*>, where *A* is a set of forbidden assignments to a subset of *V*, and *J*, called the no-good justification or culprit, is a subset of *V* such that no solution of the CSP <*V*, C_J >, given the specified domains for the variables in *V*, contains the assignments in *A*.

Example 4: Consider a CSP where *V* = {*a, b, c*}, *D*(*a*) = D(*b*) = D(*c*) = {1, 2}, with the following constraints: *C* = {C_1 ={←(*a* . 1 . *b* . 2)}, C_2 ={←(*a* . 2 . *c* . 2)}, C_3 ={←(*b* . 2 . *c* . 2)}, C_4 ={←(*a* . 1 . c . 1) }, C_5 ={←(*a* . 1 . *c* . 2)}}. Each constraint C_i trivially induces a no-good. For example, C_1 implies that <{ *a* . 1, *b* . 2}, {*a, b*}> is a no-good. Now notice that if an assignment were to include *a* . *1*, constraint C_4 would preclude *c* from taking value 1 and C_5 would preclude *c* from taking value 2. Since these are the only values in the original domain of *c*, we can infer the new constraint {←(*a* . *1*)}. Thus, the pair <*A, J*>, where *A*={*a* . *1*} is also a no-good, for some justification J. What is the justification J? The constraints that imply the no-good are C_4 and C_5: it is as a result of these two constraints that we cannot assign a the value 1. Thus, the variables that "justify" the no-good are the variables of these two constraints and so $J = \{a, c\}$. Then $C_J = \{C_2 , C_4 , C_5\}$ and, as the definition requires, $A=\{a .\ \ 1\}$cannot be part of any solution to the CSP $<V, C_J>$[10]. Notice that a no-good $<A, J>$ does not only depend on the constraints C of the CSP, but also on the domains of the variables. If the domain of c in this example contained more values than 1 and 2 we could not have inferred that $A=\{a .\ \ 1\}$ is an induced constraint.

The above example illustrates a particular point: knowing a set of no-goods, we may be able to infer other no-goods. The following two theorems present two methods for such inferences.

Theorem 4: Let $<A, J>$ be a no-good. Then $<A. J, J>$ is also a no-good, where $A .\ V$ denotes the assignment that results from projecting assignment A on the variables of V (Theorem 3.2 in [Schiex and Verfaillie 1994]).

Intuitively, the theorem states that we can reduce the assignment of a no-good, by only considering the variables in the justification. For example, if $<\{a .\ \ 1, c .\ \ 2\}, \{a, b\}>$ is a no-good, then $<\{a .\ \ 1, c .\ \ 2\}. \{a , b\}, \{a, b\}> = <\{a .\ \ 1\}, \{a, b\}>$ is also a no-good.

Theorem 5: Let A be a (partial) assignment of the variables in V, v_s be an unassigned variable in V, and $\{A_1 , \ldots, A_m\}$ be all the possible extensions of A along v_s , using every possible value of $D(v_s)$. If $<A_1 , J_1> , \ldots, <A_m , J_m>$ are no-goods, then $<A , ._i J_i>$ is a no-good (Corollary 3.1 in [Schiex and Verfaillie 1994]).

Example 5: Theorem 5 is exactly what we used intuitively in Example 4 to infer that $<\{a .\ \ 1\}, \{a, c\}>$ is a no-good. Let us illustrate now the same CSP and the same derivation again in light of Theorem 5. If we let $A = \{a .\ \ 1\}$, we see that $A_1 =\{a .\ \ 1, c .\ \ 1\}$ and $A_2 = \{a .\ \ 1, c .\ \ 2\}$ are all the possible extensions of A along variable c. Trivially (see the discussion in Example 4), $<\{a .\ \ 1, c .\ \ 1\} , \{a, c\}>$ and $<\{a .\ \ 1, c .\ \ 2\} , \{a, c\}>$ are no-goods, or equivalently $<A_1, \{a, c\}>$ and $<A_2 , \{a, c\}>$ are no-goods. By the theorem we infer that $<A, \{a, c\}>$ is a no-good too.

4.1 Building, Recording, and Using No-Goods during Search

Suppose we design our search algorithm so that, given a partial assignment A, it explores all extensions of A, and always returns one of two results: a solution, or a justification J for the

[10] The reader might wonder why we define a no-good as the pair $<A, J>$, where the justification J is the set of the *variables* involved in the constraints that imply A, instead of having J to be the set of the actual constraints. Indeed, Schiex and Verfaille [Schiex and Verfaillie 1994] record the involved constraints as the no-good justifications. For our current example, the no-good would be $<\{a .\ \ 1\}, \{C_4 , C_5\}>$ instead of $<\{a .\ \ 1\}, \{a, c\}>$. Notice that $C_J = C_{\{a, b\}}$ is a superset of $\{C_4 , C_5\}$. In general, Definition 3 leads to less specific justifications than those discovered using the Schiex and Verfaillie method. However, when employing no-goods for solving the Disjunctive Temporal Problem, it is more convenient to encode and use as justifications sets of variables than sets of constraints, especially since in the DTP the constraints are implicit). The two definitions of no-goods are equivalent for all purposes of this paper. For a more thorough discussion on the subject see [Richards 1998].

failure of all the extensions of *A*. In other words, we assume that invoking a search on the successor *A* . {*v* . u_1} returns either a solution or the no-good <*A* . {*v* . u_1}, J_1>. By Theorem 5, if all successors of *A* fail returning <*A*. {*v* . u_k}, J_k>, then we can infer the new no-good <*A*, . J_k>, which can be further reduced to the no-good <*A*.. J_k, . J_k> by Theorem 4. This no-good has a smaller forbidding assignment than all the no-goods of the successors and it can be returned recursively to the parent of the current node to explain why *A* failed to be extended to a solution. Thus, if the leaves of the search return a no-good with a justification for the failure, the internal nodes can infer and build smaller no-goods using the method just described.

The preceding discussion shows how to propagate constraints from the leaf nodes through internal nodes of the CSP. The remaining question is how to generate the no-goods at the leaves. Building no-goods at the leaves is easy for standard CSPs: if the current assignment at the leaf violates a constraint *C*, then the no-good <*A*, V_C> is returned, where V_C contains the variables in *V* that appear in the constraints in *C*. If more than one constraint *C* is violated, we can arbitrarily select one to return[11].

In the meta-CSP of a DTP, the constraints among the CSP variables are implicit and have to be inferred, and so it is not as straightforward to determine what justification to return when a constraint is violated. We distinguish two cases for when assignment *A* violates a constraint and correspondingly two ways to form a justification for the failure:

1. *A is a superset of A' for some already recorded no-good <A', J>*. In this case *J* is returned as the justification.
2. *A corresponds to an STP that is inconsistent*. Suppose that *p* is the negative cycle in the inconsistent STP. If there are no semantic branching constraints added, then this negative cycle is formed entirely from value-variable assignments in *A*. If *vars(p)* are the variables whose value assignments are the STP constraints in *p*, the set *vars(p)* is the justification that should be returned . For example, if assignment *A*={C_1 . c_{11} , C_2 . c_{21} , C_3 . c_{31}}and *p*={c_{11}, c_{21}}, then the justification *J* = {C_1, C_2} should be returned. However, if semantic branching constraints are added, then they might also participate in the negative cycle *p*, e.g. if assignment *A*={C_1 . c_{11} , C_2 . c_{21} , C_3 . c_{31} }and *p*={c_{11}, c_{21} , *v*}, where *v* is a semantic branching constraint. In this case, the set of variables that constitutes the culprit of the failure is *vars(p)* and all the variables that justify the addition of *v*. Assuming that we have a way of obtaining the justification of the semantic branching constraints, denoted by the function *just(v)* for a constraint *v*, the justification to be returned should be is *vars(p)* . $_i$ *just*(v_i), where v_i are all the semantic branching constraints that participate in the negative cycle *p*. In order to implement function *just(v)* we need to store the pairs <*v*, *J*> where *v* is a semantic branching constraint that holds in the current assignment and *J* the justification of the most recent failure prior to the addition of *v* (i.e. the failure that led to the addition of *v*).

[11] In [Schiex and Verfaillie 1994] the idea of returning more than one justification per failure is explored, but this is outside the scope of this paper.

Notice that case (1) requires that during search the current assignment *A* is checked against all recorded no-goods to determine whether *A* . *A'* for some no-good <*A', J*>. This lookup operation imposes a significant overhead for using the recorded no-goods (see [Tsamardinos 2001] for an efficient implementation of no-good lookup scheme). Recording more no-goods provides better chances for pruning the search space; however, it increases the time for the lookup operation. Thus, one needs to determine which no-goods to keep among all possible no-goods discovered during search. The easiest scheme is to limit the size of the no-goods recorded by a fixed constant *k*: a no-good assignment is recorded only if it contains less than *k* value-variable assignments (independent of the size of the justification set). In the experiments we conducted we determined the best value for *k* for the range of problems we tested, as described in Section 6.4.

5. Integrating all Pruning Methods: The Epilitis Algorithm

We are now ready to describe our algorithm for DTP solving. Called Epilitis, the algorithm combines all the pruning methods used in the previous literature on DTP solving—namely Conflict Directed Backjumping, Removal of Subsumed Variables, and Semantic Branching—and it adds in the no-good recording scheme of discussed in Section 4. The main difficulty in designing the algorithm is that no-good recording, CDB and SB interact and special attention is required to combine them. Here we present a high-level description of the algorithm shown in Figure 12; complete details, sufficient for implementation, are provided in the Appendix A.

As in the previous approaches, Epilitis attacks a DTP by attempting to solve the associated meta-CSP, searching for a consistent component STP. It takes three arguments:

A, the current assignment of values (of the form *x-y <=b*) to variables

U, the yet-to-be-assigned variables

S, the current induced STP, which is represented by a distance array, a precedence array, and a set of pairs<*v, J*>, such that *v* are the semantic branching constraints justified by the meta-level constraints involving the variables in *J*.

When Epilitis is initially invoked to solve a DTP <*V,C*>, *A* = ¬, *U* = *C*, the variable domains are initialized as in the basic DTP algorithm of Figure 1, and the distance and predecessor arrays are empty, as are the SB constraints.[12]

[12] Recall that the distance array is the all-pairs shortest path matrix, and the predecessor array stores a predecessor of *j* on the shortest path from *i* in all entries <*i,j*> that are not on the main diagonal.

```
Epilitis(A, U, S)

1. /* Removal of Subsumed Variables) */
2. For all variables x in U,
3.         Remove x from U if for any value v in d(x) the Subsumption Condition holds in the
4.           current STP S.
5. EndFor
6. If U=¬ then
7.         Stop and report A as a solution
8. Else
9.         Select a variable x in U
10.        For all values v in the current domain of x, d(x)
11.                forward-check v
12.                If forward-check fails with justification Just,
13.                        record <A. {x. v}. Just, Just> (No-good recording)
14.                Else,
15.                        Try extending A by {x. v} (Recursively call Epilitis).
16.                        If the call returns with justification J_i that does not involve x,
17.                                backjump and return J_i (Conflict-Directed Backjumping)
18.                        EndIf
19.                EndIf
20.                If value v fails, add reverse(v) to S, EndIf (Semantic-Branching)
21.        EndFor
22.        If all values v (in the original domain of x, D(x)) have failed or been removed from D(x)
23.          with justifications J_i
24.                record <A.. _i J_i, . _i J_i>, return . _i J_i (No-good recording)
25.        EndIf
26. EndIf
```

Figure 12: High-level description of Epilitis algorithm.

On any (recursive) call, if U=¬ then A represents a solution to the DTP. If any variable in U is subsumed by S, then it is removed from U. Next, a variable x in U is selected and an attempt is made to extend A by making an assignment to x. Otherwise, each value v_k in $d(x)$ is considered in turn and A is extended to $A' = A$. $\{x$. $v_k\}$, while constraint v_k is propagated in S. If forward-checking a value v_k reduces the domain of some variable to the empty set, then a dead-end has been reached. At this point, Epilitis records the no-good <A. $\{x$. $v\}$. *Just, Just*> where *Just* is the justification for the failure as discussed in 4.1. Otherwise, if forward-checking does not lead to a dead-end, then Epilitis is recursively invoked.

If a dead-end has been reached for every possible extension of x among all v_k then we build and record another no-good <A.. $_k J_k$, . $_k J_k$> where J_k are the justifications for each A . $\{x$. $v_k\}$ failing.

CDB is implemented with the following scheme: If while recursively calling Epilitis with assignment $A' = A$. $\{x$. $v_k\}$ a failure occurs with justification J, then there is no need to try another value v_p if x does not appear in J: if x is not in the culprit of the failure, the same dead-end will be encountered again for A . $\{x$. $v_p\}$. Thus, we can stop trying any remaining values in the domain of x, and backjump over x.

Finally, SB is implemented by propagating the **reverse** w=←v_k in the current STP S, when A . $\{x$. $v_k\}$ leads to failure. However, recall that in order to create the correct justifications for building no-goods and performing CDB the pairs <*w*, *J*> of the current set of semantic branching constraints need to be maintained, where *J* is the justification for adding *w*.

6. Experimental Results

6.1 Experimental Setup

We next describe the results from a series of experiments that we ran on Epilitis and the solver of Armando, Castellini, and Giunchiglia called TSAT, publicly available at http://www.mrg.dist.unige.it/~drwho/Tsat. (As described further below, TSAT has been shown to be the most efficient DTP solver previously developed [Armando, Castellini et al. 1999].) The goal of the experiments was to assess the effectiveness of various combinations of the pruning strategies described in the previous sections. As is customary in the DTP literature [Stergiou and Koubarakis 1998; Armando, Castellini et al. 1999; Oddi and Cesta 2000; Stergiou and Koubarakis 2000], experimental sets were produced using the random DTP generator implemented by Stergiou, in which DTPs are instantiated according to the parameters <*k, N, m, L*>, where *k* is the number of disjuncts per constraint, *N* the number of DTP variables, *m* the number of DTP constraints, and *L* a positive integer such that for all the disjuncts $x - y = b$, b . [0, L] with uniform probability. In the random DTP problems we used, we used the typical settings in the literature where $k = 2$, $L = 100$, and N . {10, 15, 20, 25, 30, 35}. Parameter *k* is chosen to be 2 because this is the case for constraints that typically appear in many planning and scheduling (e.g. A<B or B<A). We also employ a derived parameter *R*, the ratio of constraints over variables, *m*/*N*. For each setting of *N*, we varied *R* from 2 to 14, and we generated 50 random problems for each setting of *N* and *R*. (For example, we generated 50 problems for the case where *N* is 30 and *R* is 10; those problems have 30 variables and 300 constraints). The total number of experiment problems was 50 · 13 · 6 = 3900 (13 values for R, 6 values for *N*). The domains of the variables are integers instead of reals so that semantic branching can easily be implemented: the negation of the constraint $x - y = b$ is $y - x = -b - 1$[13]. This is again standard with the rest of the literature.

The output of Epilitis provides the following statistics for each DTP solved:

The ***Time*** it took to solve the problem.

The number of constraint checks ***CCs*** (i.e., the number times the algorithm checked the FC-condition or the Subsumption-Condition).

The number of search ***Nodes*** generated.

The number of constraint propagations ***CProps*** (i.e., number of calls to **maintain-consistency**).

[13] There are specific reasons why we chose to implement $\neg(x\text{-}y\leq b)$ as $y\text{-}x\leq\text{-}b\text{-}1$. First, if the variables are integer-valued *and* all the bounds are integer valued, then obviously y-x<-b is equivalent to y-x≤-b-1 which is stricter than y-x≤-b-ε. In addition, both in TSAT and in the Oddi and Cesta's work, this is the method that semantic branching has been implemented. Thus, it would be unfair to compare Epilitis with TSAT using any other method. If the assumption of integer valued variables and bounds does not hold, semantic branching can be implemented as y-x≤-b-ε or even y-x≤-b (which is not as strict as possible, but sound).

The number of no-goods checks ***NCs***,(i.e. the number of times a no-good is checked for retrieval).

The number of no-goods recorded ***NGs***.

In the graphs and tables showing the results below, except where otherwise noted we present the median of the above statistics over the series of the 50 experiments with the same parameters *N* and *R*. Again, this is consistent with the literature on DTP solving.

The Epilitis algorithm was implemented in Allegro Common Lisp 5.0. Both Epilitis and the ACG solver were ran on the same Intel Pentium III machine running Windows 2000, having 384MB memory and a clock speed of 1GHz. There is no time-out for the experiments run using Epilitis, but we used the time-out of 1000 seconds provided as a default with the ACG solver. This time-out limit is reasonable because it is an order of magnitude larger than the maximum amount of time taken by Epilitis to solve any of the test problems. Note, moreover, that by imposing a time-out limit on ACG but not on Epilitis, we are, if anything, providing an advantage to ACG in the experimental comparison.

All of our experiments confirm the existence of a critical region for values *R=5, 6, 7, 8*, where the percentage of solvable problems is less than 10% and the median time to find a solution or prove there is no solution to a DTP problem substantially exceeds the median time taken when R<5 or R>8.

6.2 Pruning Power of Techniques

In the first set of experiments we investigated the pruning power of all the pruning methods and combinations thereof. This set of experiments answers the following questions: (i) can all pruning methods be integrated efficiently? (ii) how do the pruning methods and their combinations compare quantitatively?

The pruning methods we tried are:

Removal of Subsumed Variables (***RSV***)

Conflict-Directed Backjumping (***CDB***)

Semantic Branching (***SB***)

No-good Recording (***NG***)

In Epilitis all of the above methods can be individually turned on and off, providing us with the opportunity to try any combination we desire. The only limitation is that whenever *NG* is on, *CDB* must also be turned on. We name our graphs and tables using the following convention: we list the options that were turned on separated by spaces or dashes. When we bound the size of the no-goods, as explained in Section 4.1, we follow the name with the numerical bound. For example, CDB-RSV-SB-NG-10 is Epilitis with CDB, RSV, SB, NG on and a maximum size of no-goods set to 10, and CDB-RSV-SB-NG the same algorithm with no bound on the size of no-goods. We use the term "Nothing" to identify "bare" Epilitis, with no pruning techniques turned on.

For this set of experiments we used the following ***dynamic*** variable and value heuristics:

Select the variable according to the MRV (*Minimum Remaining Values*). Break the ties by selecting the variable that contains the value that *maximizes* the number of pair-wise inconsistencies with the values in the domains of the unassigned variables.

Select the value that *minimizes* the number of pairwise inconsistencies with the values in the domains of the unassigned variables.

This heuristic is typical in the CSP literature. The idea is that by choosing the variable with the value that maximizes the pairwise inconsistencies, the branching factor of the search is reduced, since this is the variable that most constrains the search. On the other hand, when we select a value we prefer the one that least constrains the search so that we increase the probability of finding a solution that contains this value.

Ratio	nothing	rs	cdb	cdb rsv	ng_10	ng	sb	sb rsv	cdb sb	cdb sb rsv	cdb sb rsv ng	cdb rsv sb ng 10
2	0.02	0.02	0.02	0.02	0.03	0.03	0.02	0.02	0.02	0.02	0.03	0.021
3	0.05	0.05	0.05	0.05	0.06	0.06	0.05	0.05	0.05	0.05	0.06	0.06
4	0.14	0.13	0.13	0.11	0.14	0.15	0.14	0.12	0.14	0.11	0.13	0.13
5	1.91	1.39	1.05	0.811	0.49	0.551	0.531	0.51	0.501	0.451	0.421	0.431
6	4.1	3.33	2.8	2.39	1.53	1.46	1.43	1.34	1.25	1.24	1.04	0.941
7	1.93	1.74	1.87	1.5	1.07	1.31	1.05	1.01	0.981	0.971	1.02	0.851
8	1.15	1.11	1.05	0.892	0.781	0.982	0.671	0.651	0.661	0.621	0.751	0.701
9	0.711	0.661	0.671	0.611	0.621	0.681	0.551	0.54	0.521	0.53	0.62	0.571
10	0.671	0.611	0.631	0.571	0.6	0.66	0.55	0.551	0.541	0.511	0.571	0.531
11	0.56	0.551	0.521	0.521	0.541	0.61	0.461	0.441	0.451	0.441	0.531	0.511
12	0.491	0.501	0.481	0.461	0.491	0.551	0.451	0.431	0.43	0.431	0.521	0.48
13	0.461	0.48	0.461	0.461	0.461	0.521	0.45	0.44	0.42	0.411	0.51	0.48
14	0.441	0.42	0.441	0.43	0.471	0.541	0.42	0.41	0.421	0.411	0.551	0.491

Table 2: The ordering of the pruning methods according to Median Time when R=6, and N=20.

Ratio	rsv	cdb	cdb rsv	ng_10	sb	ng	cdb sb	sb rsv	cdb sb rsv	cdb rsv sb ng_10	cdb sb rsv ng
2	0.04	0.04	0.05	0.04	0.04	0.05	0.04	0.04	0.04	0.05	0.05
3	0.1	0.11	0.111	0.12	0.11	0.12	0.11	0.1	0.1	0.11	0.11
4	0.37	0.291	0.31	0.32	0.361	0.311	0.311	0.32	0.251	0.29	0.281
5	9.22	4.3	2.41	1.06	2.08	0.892	1.5	1.69	1.04	0.811	0.681
6	40.8	27.5	24.8	10.1	8.77	8.72	7.98	7.7	7.43	7.05	5.38
7	18.5	16.3	14.1	8.56	7.72	7.66	7.94	7.47	6.94	6.78	7.09
8	6.8	5.81	5.02	4.1	3.37	3.83	3.36	3.2	3.14	2.74	2.71
9	4.99	4.46	4.45	3.51	3.3	3.56	3.14	3.1	2.89	2.83	2.36
10	3.46	2.69	2.45	2.08	2.01	2.36	1.94	1.97	1.74	1.92	1.9
11	2.48	2.21	1.86	1.9	1.52	2.17	1.57	1.42	1.36	1.61	1.52
12	2.44	2.3	1.98	1.71	1.44	1.86	1.58	1.36	1.36	1.53	1.44
13	1.84	1.63	1.44	1.31	1.26	1.48	1.2	1.13	1.11	1.18	1.16
14	1.68	1.3	1.25	1.38	1.18	1.51	1.07	1.12	1	1.3	1.22

Table 3: The ordering of the pruning methods according to Median Time when R=6, and N=25. In Table 2, Table 3, and Table 4 we show the results for N=20, N=25, N=30 for various pruning

methods and their combinations. The results for N<20, not reported here, are similar. The columns are listed in increased order of efficiency for Ratio=6; this is the peak of the critical region. The tables are not complete, i.e., some pruning method combinations are missing, because they caused the algorithm to be too slow for the experiments to complete (e.g. "nothing", i.e. the no pruning methods version is not reported in Table 3). All times are reported in seconds, as in all experiments in this paper. We selected size 10 for bounding the no-good size because in other experiments (described subsequently), size 10 was determined to be optimal size for the Epilitis with all pruning methods on. Although it would be desirable to have an analytic technique for predicting the optimal size of no-goods, we do not know of a suitable such account, and to date, all results on optimal no-good size have been determined experimentaly.

As expected in Tables 2-4, "nothing" performs the worst, then RSV, then CDB, then SB, NG, and NG-10 following closely together. It makes sense to compare the time of each algorithm, since their underlying implementation is the same.

Ratio	sb	cdb sb rsv	cdb sb	sb rsv	cdb sb rsv ng_10
2	0.07	0.07	0.07	0.07	0.08
3	0.17	0.18	0.17	0.17	0.18
4	0.4	0.39	0.371	0.36	0.39
5	10.3	7.42	7.12	8.25	4
6	149	142	140	138	79.8
7	74.6	70.5	69.7	78.2	48.8
8	29.4	26.6	25.9	31.5	21.1
9	13.9	12.6	12.4	14.1	11.1
10	9.73	9.15	8.92	10.3	7.72
11	6.73	6.28	6.09	6.69	4.93
12	4.47	4.64	4.42	4.61	4.12
13	4.32	4.31	3.77	4.38	3.97
14	3.96	4.02	3.91	4.17	3.2

Table 4: The ordering of the pruning methods according to Median Time when R=6, and N=30

Since the search space increases exponentially with the number of variables, the results become more significant as *N* grows. Thus the differences among pruning methods is most obvious in Table 4, which shows the results for *N*=30. The worst combination is the CDB-RSV: we were not even able to complete this experiment for N=30. The best performance is the CDB-SB-RSV-NG-10. When we did not bound the size of the no-goods, the performance of the algorithm was seriously degraded for N=30 and this is why it is not included in the table.

We now compare the different pruning methods strictly according to their pruning power, i.e. not including the computational overhead to implement them. Table 5 shows the statistic $Nodes_c/Nodes_{Nothing}$ where $Nodes_c$ is the median number of search nodes explored by the algorithm in each column *c,* and $Nodes_{Nothing}$ the search nodes explored by Epilitis with no pruning methods. As we would expect, the more methods we add, the more we prune the search space. In this case NG is the best single[14] method, exploring only 27.31% of the whole search space (i.e. when no pruning method is on) for R=6. NG is even better than the combination of all three other methods CDB-SB-RSV, which explores 32.52% of the space. This result encourages us to look for even more efficient implementations of recording and retrieving no-goods to reduce the overhead of the technique. Table 6 supports the same argument, showing the effect of pruning methods on the search space explored for N=30, where the search space is significantly larger than for N=20 (the value in Table 5). The statistic displayed is $Nodes_c/Nodes_{SB}$, where $Nodes_c$ is the median number of search nodes explored by the algorithm in each column *c,* and $Nodes_{SB}$ the search nodes explored by Epilitis with SB on. For example, in the last column, for R=6, we see that the ratio is 38.99% meaning that the algorithm CDB-RSV-SB-NG-10 explored only 38.99% of the space the algorithm SB explored on problems for N=30 and R=6. The results show in an impressive way the pruning power of no-goods: the last column, corresponding to the algorithm with the no-goods on, display a significant reduction of the space searched.

Ratio	**rsv**	**cdb**	**cdb rsv**	**sb**	**sb rsv**	**cdb sb**	**cdb sb rsv**	**ng_10**	**ng**	**cdb sb rsv ng_10**	**cdb sb rsv ng**
2	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
3	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
4	100.00%	100.00%	98.91%	97.83%	97.83%	97.83%	97.83%	100.00%	100.00%	97.83%	97.83%
5	76.00%	42.00%	42.00%	30.40%	30.40%	27.73%	27.73%	20.33%	17.53%	15.67%	15.13%
6	88.84%	67.36%	63.64%	35.29%	35.29%	32.77%	32.52%	31.12%	27.31%	19.75%	19.75%
7	89.71%	79.22%	73.69%	50.49%	50.49%	45.24%	44.66%	39.13%	39.13%	31.17%	32.82%
8	100.00%	84.01%	77.07%	54.72%	54.72%	51.25%	50.67%	52.99%	53.95%	44.51%	44.12%
9	91.20%	84.00%	84.00%	78.00%	78.00%	72.00%	72.00%	70.80%	70.00%	59.60%	59.60%
10	95.24%	81.43%	78.10%	77.62%	77.62%	69.05%	69.05%	68.10%	68.10%	56.19%	56.19%
11	98.57%	81.43%	81.43%	77.14%	77.14%	67.86%	67.86%	71.43%	71.43%	66.43%	66.43%
12	100.00%	96.12%	96.12%	95.15%	95.15%	86.41%	86.41%	84.47%	84.47%	78.64%	78.64%
13	100.00%	88.64%	88.64%	92.05%	92.05%	81.82%	81.82%	84.09%	84.09%	79.55%	79.55%
14	100.00%	97.56%	97.56%	95.12%	95.12%	87.81%	87.81%	82.93%	82.93%	82.93%	82.93%

Table 5: The statistic Median Nodes divided by Median Nodes of "Nothing" for N=20. The pruning methods are sorted according to this statistic for R=6.

Summarizing the results of this section:

A rough partial ordering of the pruning methods is RSV < CDB < CDB-RSV < NG-10 < SB < {CDB-SB, SB-RSV, CDB-SB-RSV} < CDB-SB-RSV-NG-10.

[14] Recall, however, that when NG is on, CDB is also on, so the comparison is not entirely fair.

No-good learning needs to limit the size of the no-goods recorded because asymptotically the overhead of recording and looking-up all the possible no-goods greatly outweighs the benefits.

SB is the best single pruning method in terms of performance, i.e. displaying a good trade-off between pruning power and implementation overhead.

NG is the best single pruning method in terms of pruning, even better than all the other methods combined CDB-SB-RSV.

The Epilitis with options CDB-SB-RSV-NG-10 considerably improves performance over all other methods combined CDB-SB-RSV.

Ratio	sb rsv	cdb sb	cdb sb rsv	cdb sb rsv ng 10
2	100.00%	100.00%	100.00%	100.00%
3	100.00%	100.00%	100.00%	100.00%
4	89.31%	89.94%	89.31%	89.31%
5	92.96%	71.37%	69.25%	32.16%
6	100.00%	94.93%	94.93%	38.99%
7	100.00%	97.13%	93.03%	47.13%
8	100.00%	89.59%	89.59%	55.90%
9	100.00%	86.26%	86.26%	66.79%
10	100.00%	93.37%	93.37%	65.06%
11	100.00%	85.98%	85.98%	60.31%
12	100.00%	92.95%	92.95%	63.57%
13	100.00%	95.23%	95.23%	72.57%
14	100.00%	96.97%	96.97%	66.67%

Table 6: The statistic Median Nodes divided by Median Nodes of SB for N=30. The pruning methods are sorted according to this statistic for R=6.

6.3 The Number of Forward-Checks is the Wrong Measure of Performance

Our first set of experiments were designed to compare the effectiveness of alternative combinations of pruning strategies and it was straightforward to present the results of those experiments since what we are concerned with is precisely the number of search nodes in the meta-CSP that are pruned. In the third major experiment, which we present below in Section 6.5, we compare Epilitis running with the most effective combination of pruning heuristics, to TSAT, the previous most effective DTP solver. It is less obvious what metrics to use in this comparison. It is customary in the literature to compare DTP solvers and report their performance using the number of forward-checks—more precisely the total number of values that the algorithm forward-checked during search, also called consistency checks *CCs*[15]

[15] In our implementation a consistency-check is essentially checking the FC-condition. We also felt that we should count as a consistency-check determining whether the Subsumption-Condition holds, because both are similar operations having similar functions and take the same time. Not counting the Subsumption-Condition checks in *CCs* would favor all algorithms with RSV on since those are the ones that perform this operation.

[Stergiou and Koubarakis 1998; Armando, Castellini et al. 1999; Oddi and Cesta 2000; Stergiou and Koubarakis 2000]. Such comparisons make the implicit hypothesis that the number of consistency checks is a machine and implementation independent measure of performance. In this section we present both theoretical arguments and empirical evidence that this hypothesis is false and CCs is the wrong measure of performance.

There are at least three reasons for rejecting CC counts as a performance metric. First, the use of no-good recording in Epilitis gives an unfair advantage for a comparison based on CC counts. This is because no-good recording requires significant overhead to record and retrieve no-goods, and this overhead is not represented in the number of forward-checks.

Second, as discussed in Section 2.2, the time required for each consistency check depends on the method used for maintaining consistency. For example, when the distance array of each current STP is available, checking the FC-condition takes constant time, but when it is not, more time might be required.

The third argument against the use of CC-counts as a metric is that there are techniques that have been employed in previous DTP solvers that result in fewer values being forward checked even though more time is spent exploring. For example, a technique used by Stergiou and Koubarakis [Stergiou and Koubarakis 1998; Stergiou and Koubarakis 2000] and ACG [Armando, Castellini et al. 1999] is what we will call *Forward-Checking Switch-off* (FC-Off). In Appendix C, we provide an example that illustrates that that FC-Off can reduce the number of forward-checks by increasing the number of search nodes explored and thus it may even decrease the performance of a DTP solving algorithm.

As a result of the three problems with using *CC* counts as a measure of performance, we report actual computation times used in our comparison of TSAT and Epilitis. One drawback of such a comparison is that we cannot as easily draw conclusions about the pruning efficiency of Epilitis' additional pruning methods, such as RSV, CDB, and NG, since TSAT uses a very inefficient method for consistency checks. Thus, the better performance of Epilitis might be attributed only to the better consistency checking techniques it employs. We cannot totally dismiss this hypothesis until a version of TSAT is re-implemented using better forward-checks methods. However, our first set of experiments, which analyzed the pruning methods and showed their effectiveness, make it unlikely that efficient consistency checking is solely responsible for Epilitis' performance advantage.

6.4 Heuristics and Optimal No-good Size Bound

Before presenting the actual comparison of Epilitis and TSAT, we need to pin down one more detail, namely, the search heuristic used for selecting which variable/value combination to select during each stage of the search. Interestingly, the use of no-good recording increases the range of search heuristics available, because the heuristic itself can take into account the no-good information. In turn, however, this means that the performance of the search heuristic is intertwined with the size of the no-good recorded. Thus, we designed a factorial experiment aimed at discovering the best combination of search heuristic (from amongst a set of plausible heuristics) and bound on no-good size.

As heuristic information we considered four functions that estimate how much a value x constrains the remaining search space. These are:

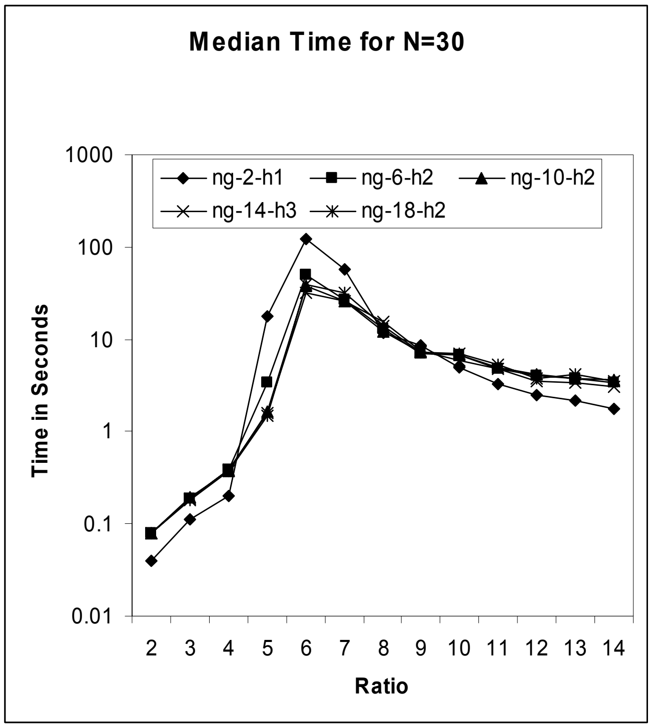


Figure 13: Comparison for the best overall algorithm for N=30.

> ***E0***: the number of remaining values that are pairwise inconsistent with x (i.e. calculated dynamically during search using the current domains).
>
> ***E1*** : the number of values that are pairwise inconsistent with *x* determined *statically* before search begins.
>
> ***E2*** : the number of the remaining values that are pairwise inconsistent with *x* plus the number of no-goods *x* appears in.
>
> ***E3*** : the number of the remaining values that are pairwise inconsistent with *x*. Ties are broken by the number of no-goods *x* appears in.

As estimators of how much a variable *v* constrains the remaining search space we used the maximum of the value estimator used over all remaining values in *v*'s domain.

In each of our heuristics we follow the principle of selecting the variable that most constrains the remaining search space, in an attempt to minimize the branching factor. Thus, a variable with minimal current domain is chosen first (*Minimum Remaining Values* heuristic) by default. However, since all domains have maximum size two, it is often the case that there are many ties, and these are then broken by using one of the above estimators *E0-E3*, giving rise to the four heuristics ***H0-H3*** respectively.

In contrast, as value selection principle we select the value that least constrains the search space, in an effort to hit a solution faster. We again use the estimators above. For example, in ***H2*** we would first select the variable *v* with the value that maximizes the number of pairwise inconsistencies and participates in the most recorded no-goods, among all the variables with least domain size. Then, we would select the value in *v*'s domain that minimizes this estimator (***E2***).

Figure 13 presents the results. The *x*-axis shows the R, ratio of constraints to variables; this is the critical parameter for the DTP-solving problems. The *y*-axis shows computation time taken, in seconds; note that the scale is logarithmic. We name the curves as "NG *x y*" to denote Epilitis with *all the pruning options on,* where *x* is the limit on the size of no-goods, and *y* is the search heuristic used. We show only the graph for N=30 since this is the largest size we ran. For each no-good size the best heuristic is selected (e.g. for size 6 we determined the best heuristic to be ***H2***). Overall, the combination of ***H2*** with size bound of 10 works best, although ***H3*** with bound of 14 come very close. However, we suggest using ***H2*** with size bound of 10 because it exhibits a better average case behavior.

To summarize the results of this section:

The recorded no-goods do not only prune the search space but can also be effectively used as heuristic information.

Epilitis with CDB, SB, RSV, NG on, maximum no-good size 10, and heuristic ***H2*** is the best algorithm in the set of experiments we ran.

6.5 Comparing Epilitis to the previous state-of-the-art DTP solver

In Section 7 below we provide details of Epilitis' three predecessors: one developed by Stergiou and Koubarakis [Stergiou and Koubarakis 2000], one by Oddi and Cesta [Oddi and Cesta 2000], and one by Armando, Castellini, and Giunchiglia (TSAT) [Armando, Castellini et al. 1999]. Oddi and Cesta present an experimental comparison, noting that while their algorithm consistently outperforms that of Stergiou and Koubarakis, it is at best competitive with TSAT algorithm. Moreover, they show that TSAT is particularly good at the hardest problems, i.e., those in the critical region. As Oddi and Cesta note, "further work [on their system] will be needed to clearly outperform TSAT." These are the problems that are most significant, since problems outside of this range can already be solved very quickly. Given these results, we view TSAT as the state-of-the-art predecessor to Epilitis, and conduct head-to-head experiments with it. It is worth noting, however, that there is one class of problems for which Oddi and Cesta's approach outperforms that of TSAT: problems with small *R* values ($R = 5$). On these problems, Oddi and Cesta's system is about one order of magnitude faster than TSAT. Although we have not conducted a head-to-head comparison of Epilitis with Oddi and Cesta's system on such problems, Epilitis is also about an order of magnitude faster than TSAT, and thus it is reasonable to conclude that it is competitive with Oddi and Cesta's system for these (relatively easy) problems.

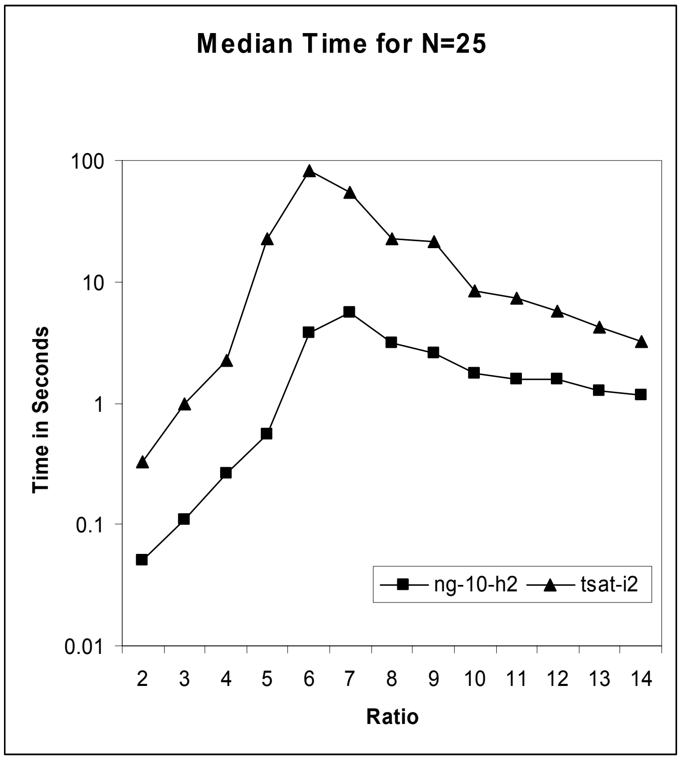
We now turn to the details of the final experiment, in which we compare CDB-SB-RSV-NG-10-H2 with TSAT. Figure 14 shows the results for N=25 and N=30. As explained above, we report overall computation time; as in Figure 13, the *x*-axis shows R, and the *y*-axis, which is logarithmic, shows median computation time in seconds. Epilitis is faster by about two orders of magnitude for the larger (N=30) problems. Also recall that TSAT has a time-out of 1000 seconds imposed, and so the real median time taken by their program might be significantly more than is indicated here. For example, for N=30 and R=6 the median time is exactly 1000 seconds implying that the TSAT solver timed-out on more than half the problems.

We also ran the best version of Epilitis on problems of up to size N=35 (the largest size of random DTP problems reported as so far) and we observed that the algorithm scales relatively well. Figure 15 shows the performance of Epilitis on problems of different sizes. For the larger problems where N=35 Epilitis has a median time performance of about 100 seconds, while the corresponding TSAT performance is more than 1000 seconds for problems of size N=30. The overall performance of TSAT is also shown in the same figure. As we can see TSAT requires about one order of magnitude more time every time N is increased by 5. In contrast Epilitis' performance does not degrade as fast.

Summarizing the results of this section:

Epilitis is almost two orders of magnitude faster than the previous state-of-the-art DTP solver TSAT on standard benchmark problems.

Epilitis' performance scales comparatively well as the size of the problems increase.



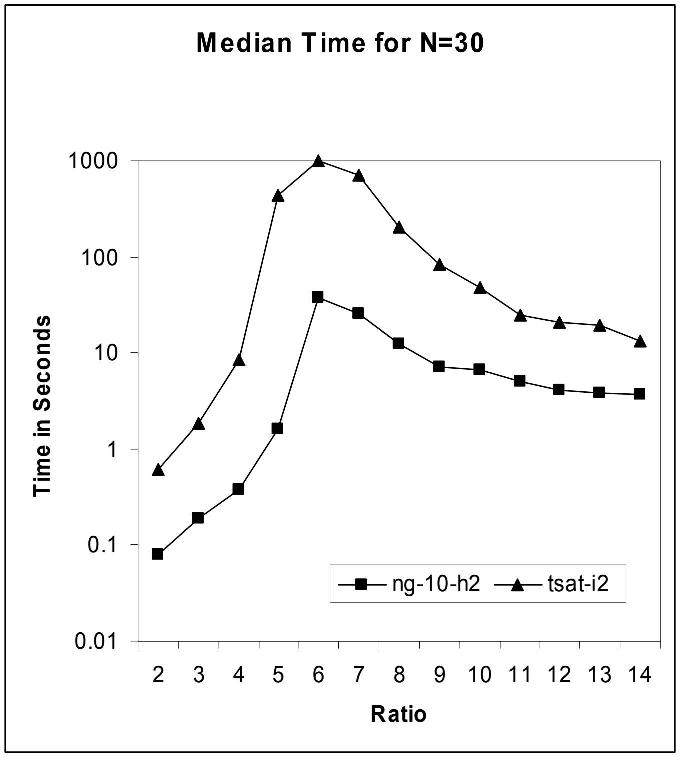
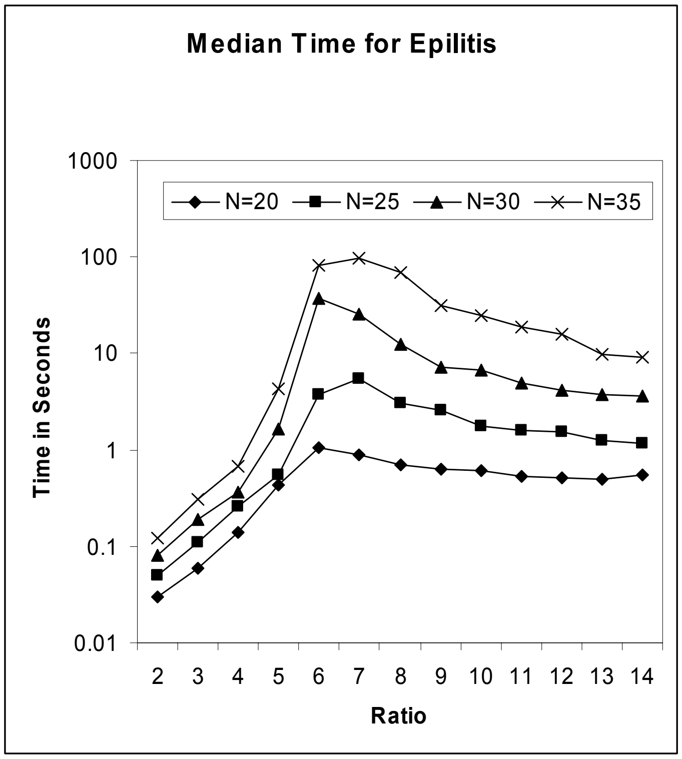


Figure 14: Comparison of Epilitis and ACG's solver for N=25 and N=30.



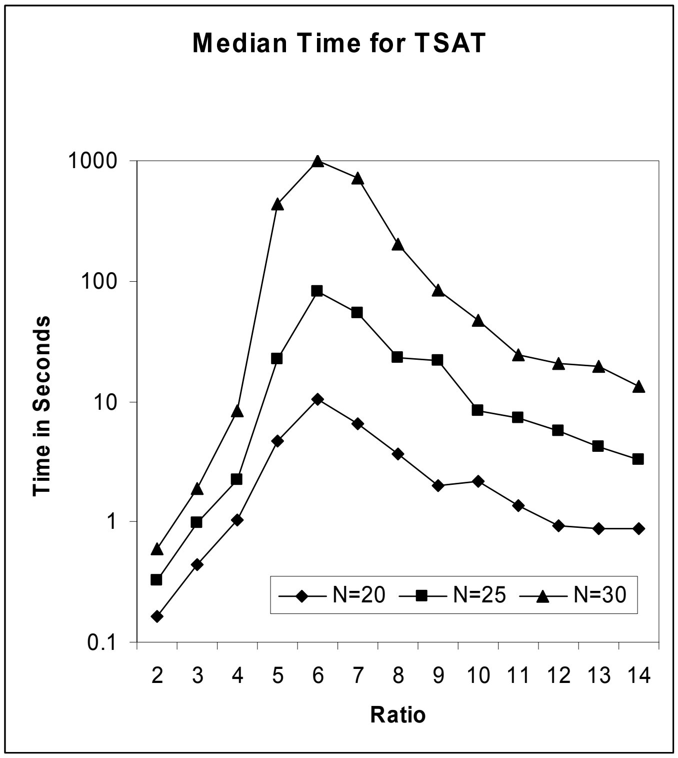


Figure 15: The median time performance of Epilitis and TSAT.

6.6. Application Experience

The previous sections have described controlled experiments we performed using synthetic, abstract test problems, to analyze the performance of the various pruning strategies developed for DTP solving and to compare Epilitis with the previous state-of-the-art system. We also have some experience using Epilitis in a large application, which we briefly describe here.

Autominder [Pollack 2002] is an intelligent cognitive orthotic system: a system designed to help older adults with memory decline by providing them with adaptive, personalized reminders about their daily activities. Autominder has three main components: a Plan Manager, which stores a plan of it's clients daily activities, and is responsible for updating it and identifying potential conflicts in it; a Client Modeler, which uses information about the client's observable activities[16] to track the execution of the plan, inferring what activities the client has already performed and what activities are pending; and a Personal Cognitive Orthotic, which reasons about any disparities between what the client is supposed to do and what she is doing, and makes decisions about when to issue reminders.

The relevant component for the current paper is the Plan Manager. It maintains a model of the client's daily plan encoded as a DTP. It then invokes Epilitis to update the plan in response to four types of events:

(1) The addition of a new activity to the plan.
(2) The modification or deletion of an activity in the plan.
(3) The execution of an activity in the plan, as reported by the Client Modeler.
(4) The passage of a time boundary in the plan.

In each of these cases, the Plan Manager formulates a DTP: for instance, when a new activity is added, the DTP consists of the constraints in the original client plan, the constraints in the new activity, and a set of constraints generated to represent the resolution of any conflicts between the two. Epilitis then attempts to solve the DTP, indicating whether or not it succeeded, and returning any newly required constraints. For instance, the addition of an activity may result in added constraints on the time of performance of a previously added activity.

In general, the size of the plans managed by Autominder are small by the standards of the planning community. (On the other hand, our focus is not on plan generation, but on a range of other tasks, such as plan monitoring, update, and dispatch.) Generally the client plan has on the order of 30 actions, meaning that there are 60 events (start and end points), and, with typically temporal constraints, the representation requires about 4 or 5 constraints per action, and about 2 or 3 disjuncts per constraint. For problems of this scale, Epilitis nearly always produces solutions (or determines inconsistency) in less than a second, an amount of time that is well within the bounds we require.

[16] The current version of Autominder is deployed on a mobile robot, and uses on-board sensors to observe a client's movement about her home.

Technique	SK	ACG	OC	Epilitis
		Temporal Reasoning		
Constraint Propagation	Maintain IDPC, O($\|V\|^2$)	N/A	Maintain IFPC, O($\|V\|^2$)	Maintain IFPC, O($\|V\|^2$)
Forward-Checking (time complexity is for each value) [17]	Find distance, check FC-Condition. Actual implementation O($\|V\|^2$); could be done in O($\|V\|$).	Run an STP consistency checking algorithm, Actual implementation: O($2^{\|V\|}$). Could be done in O($\|V\|^3$).	Lookup distance, check FC-Condition, O(1)	Lookup distance, check FC-Condition, O(1)
Value Subsumption (time complexity is for each value) [18]	Not in the original implementation. Could be done by finding distance, then checking Subsumption-Condition, O($\|V\|$)	Not in the original implementation. Could be done by running an STP consistency algorithm O($\|V\|^3$).	Lookup distance, check Subsumption-Condition, O(1)	Lookup distance, check Subsumption-Condition, O(1)
		Searching Methods		
CDB	Yes	No	No	Yes
RSV	No	No	Yes	Yes
SB	No	Yes	Yes	Yes
FC-off	Yes	Yes	No	Yes
NG	No	No	No	Yes
IFC[19]	No	No	Yes	No
		Heuristics		
Variable	MRV	Max-Inc	MRV	See Section 6.3
Value	The value with the time-points with the most appearances in other values	The value with the most pairwise inconsistencies (but it might be negated)	No specific heuristic	See Section 6.3

Table 7: Comparison of all DTP solvers

[17] The time complexity shown is for implementing forward checking with the best known algorithm: Given that in the SK approach STP , the current STP is in directional path consistency, we can find the distance between a pair of nodes in time O($|V|$) and check the FC-Condition. As already mentioned, in the actual implementation, SK used a less efficient scheme that takes quadratic time. In the ACG approach, we have to run an STP consistency checking algorithm, while in the actual implementation ACG use Simplex.

[18] Neither SK nor ACG use RSV, so these two cells do not refer to their actual implementation. Instead, this is the most efficient way they could have implemented RSV had they desired to, given the way they perform temporal propagation.

[19] IFC refers the to Incremental Forward Checking technique of [Oddi and Cesta 2000] in which a value $v : x\text{-}y = b_{yx}$ is forward-checked only if the distance d_{xy} has changed since last forward checking took place. If it has not, then v satisfies the FC-Condition for sure. We mention this technique for completeness. IFC does not reduce the search space and it can be used in conjunction with any pruning technique.

7. Related Work

7.1 Previous DTP Solvers

There are two previous DTP-solvers that treat component-STP selection as CSP problems and perform a search in the same meta-CSP as Epilitis: that of Stergiou and Koubarakis [Stergiou and Koubarakis 1998; Stergiou and Koubarakis 2000] (hereafter **SK**) and of Oddi and Cesta [Oddi and Cesta 2000] (hereafter **OC**). We can compare these approaches in terms of the implementation of (1) **maintain-consistency**, (2) **forward-check**, (3) the variable ordering heuristic, (4) the value ordering heuristic, (5) and the techniques employed for the search and for pruning the search space. An additional approach, which casts DTP-solving in terms of SAT, is discussed in Section 7.1.2.

7.1.1 The CSP Approaches

In the SK approach, function **forward-check** is implemented by adding a value to the current STP S and propagating using again the IDPC algorithm, identifying the inconsistency if there is one, and then retracting the constraint using again IDPC so as to be ready to forward check the next value. This requires two calls to IDPC with $O(|V|^2)$ in the worst-case for each value to be forward checked. There is of course a much faster algorithm: checking if the FC-Condition holds. The FC-condition requires the distance between two time points, which given that the SK approach maintains the current STP in directed path consistency form, can be found in $O(|V|)$ time in the worst case with the algorithm described at [Chleq 1995].

The variable ordering heuristic in SK is the *Minimum Remaining Values* (***MRV***) in which the variable with the fewest remaining values in its domain is selected first. Ties among variables are broken by choosing the variable that contains the time-points that appear the most in the rest of the variables. Each variable may contain many disjuncts and each disjunct contains two time-points, so we can choose the variable that contains the time-point with the maximum appearance in other variables/disjunctions or the variable with the largest sum of appearances of its time-points in other variables/disjunctions. The SK paper does not discuss exactly how the selection is performed. The value ordering heuristic is the same as the tiebreaker heuristic above. The disjunct whose time-points appear in the most in other variables is selected first. SK experimented with the anti-heuristic but with disappointing results.

For the approach of OC, the table summarizes well the design-choices made. The OC variable ordering heuristic is the MRV with no other tie-breaking heuristics. There is no particular value ordering heuristic.

7.1.2 The SAT Approach

The Armando, Castellini, and Giunchiglia (ACG) approach differs from the previous ones in that it treats the component-STP selection not as a meta-CSP problem, but as a SAT problem instead. However, we can still use the above classification scheme to compare the approach: The ACG algorithm does not use **maintain-consistency**. Every time the algorithm requires checking the consistency of a set of STP-like constraints they use a version of the Simplex algorithm for linear programming. Simplex has exponential worst-case performance.

During a preprocessing step, ACG enhances the SAT formula with clauses (equivalently variables in a CSP-based approach) that correspond to inconsistent pairs of literals (equivalently values in a CSP-based approach). This provides additional guidance to an MRV-like criterion for variable selection: they choose the clause that contains the literal with the greatest number of occurrences in the clauses of minimal-length (which implies they will choose the clause with the literal that participates in the most pairwise inconsistencies with other literals). We will call this heuristic ***Max-Inc*** because essentially it picks the clause with the literal that participates in most pairwise inconsistencies.

For variable ordering, they choose the literal that maximizes the number of pairwise inconsistencies in the previous step. Notice here however, that a SAT-based procedure can either choose to branch on the literal c_{ij} or the literal $\leftarrow c_{ij}$. Instead, a CSP-based approach can only branch on c_{ij} , i.e. assign a value to a variable, and never the negation of the value to a variable. From the ACG paper it is unclear which branch (i.e. the positive or the negative) is taken first and how this choice is made.

Table 7 summarizes the above discussion and comparison between the different approaches. The DPT solving approaches are ordered chronologically according to the date of appearance in the literature.

7.2 Other Temporal Reasoning formalisms

The focus of this paper has been on developing more efficient techniques for solving DTPs. One question we must address is why we want to use DTPs, when there are other formalisms for temporal reasoning that are computationally more tractable. For example, as we noted at the beginning of the paper, DTPs subsume both Simple Temporal Problems (STPs) and Temporal Constraint Satisfaction Problems (TCSPs). STPs allow only non-disjunctive constraints, while TCSPs allow constraints of the form c_{i1} . .... c_{in} where each c_{ij} is of the form $x - y = b$ with the restriction that x and y are the same in all c_{ij} . STPs can be solved in polynomial time. Although the same is not true for TCSPs, they are still computationally more tractable than DTPs, because all their constraints are binary, involving only two variables, while the DTP constraints may be non-binary, and it is significantly easier to calculate path-consistency in networks of binary constraints than in networks where the constraints are non-binary [Bessiere and Regin 1997; Bessiere 1999; Bessiere, Mesequer et al. 1999].

It turns out, however, that the limitations of TCSPs do result in weak expressive power: in particular, we consider the most serious disadvantage of TCSPs to be their inability to express the fact that two intervals should not overlap. This kind of constraint is essential in scheduling and planning applications where it is often the case that some actions should not overlap, for example if they utilize the same unary resource. As was illustrated in the example in the Introduction, it is straightforward to encode such a restriction with a DTP constraint: if denote with A_S (A_E) and B_S (B_E) the start (end) times of actions A and B , then the fact that they cannot overlap can be written as the DTP constraint:

$$A_E - B_S = 0 \ . \ \ B_E - A_S = 0 \ .$$

This constraint involves four time-points A_S, B_S, A_E, and B_E and so it cannot be represented by a TCSP, but it is perfectly acceptable in a DTP. There are other, *binary*, representations however, that allow such constraints to be represented. These include the *Point-Interval-Algebra* (PIA) described in [Meiri 1991]. In PIA the variables can be either time-points or intervals; and all relations are binary: interval-interval, point-point, interval-point. The constraints between time-points can be metric TCSP constraints, while the rest of constraints are qualitative. Having two more interval variables A_I and B_I representing the intervals associated with the actions can then encode the above situation by imposing the disjunctive constraint:

$$A_I \;\{\textit{before, after}\}\; B_I$$

Although PIA can thus model prohibited overlaps, it cannot readily handle requirements of temporal separation between actions. For example, suppose that A and B are two medical treatment procedures applied to the same patient with the constraint that if A is applied first, B can only be applied 3 days later, while if B is performed first then A can be performed 2 days later. The constraint cannot be represented in PIA but written as the DTP constraint

$$A_E - B_S = -3 \;.\; B_E - A_S = -2$$

Nevertheless, extensions of the PIA have appeared that allow constraints of this sort to be represented while remaining within the realm of binary constraints [Cheng and Smith 1995; Cheng and Smith 1995].

The above argument may suggest that we can avoid non-binary constraints if we employ both intervals and time-points as our representational elements. However, there are other types of constraints that are inherently non-binary such as conditional constraints of the form "if *constraint1* then *constraint2*", e.g. "if treatment A does not last enough, then perform treatment B for at least e days." The constraint can be written as:

$$\leftarrow(\, d = A_E - A_S \,) \Rightarrow (e = B_E - B_S), \text{ or equivalently}$$
$$(\, d > A_E - A_S \,) \;.\; (e = B_E - B_S)$$

and these are only expressible with DTPs.

There are two other formalisms that are as expressive as DTPs, namely the Generalized Temporal Network (GNC) described in [Staab 1998] and the Temporal Constraint Networks (TCN) of Barber [Barber 2000]. The former is essentially a DTP-like formalism that allows conjunctions of STP-like constraints in each disjunct. Because it is straightforward to convert a constraint of this form into a standard DTP-constraint, the advantages of GNCs is not obvious. Barber's TNCs are also as expressive as DTP. A TNC is a TCSP with the addition of what Barber calls I-L-Sets. I-L-Sets (Inconsistent-Label-Sets) are essentially no-goods: an I-L-Set looks has the form $\leftarrow(c_{ij} \;.\; \ldots \;.\; c_{mn})$ denoting that the conjunction does not hold in the TCSP. A constraint $a \;.\; b$, where a and b are STP-like constraints of the form $x - y = b_1$ and $w - z = b_2$ (involving two pairs of different variables) cannot be represented as a TCSP constraint, but it can be encoded as the I-L-Set $\leftarrow(\leftarrow a \;.\; \leftarrow b)$ (using De'Morgan's rule for Boolean Algebra). However, TCNs require that the disjuncts in an I-L-Set participate in some other TCSP constraint. Thus, it is not enough to just add the above I-L-Set; we also have to add TCSP constraints so that a and b appear in the underlying TCSP. For example, we can add the TCSP constraints $(a \;.\; \leftarrow a)$ and $(b \;.\; \leftarrow b)$. By using this scheme, TNCs reach the expressiveness of the DTP, albeit in a peculiar way. To solve TCNs, Barber in [Barber 2000] provides a path-1 consistency algorithm that in essence calculates the full set of all no-goods (whose number is exponential to the number of disjuncts), but no experimental results are provided.

7.2 Scheduling algorithms

Another related area of work is that of Automated Scheduling. In particular, the Precedence Constraint Posting (PCP) technique of Cheng and Smith [Cheng and Smith 1995; Cheng and Smith 1995] bears certain similarities to DTP solving. Cheng and Smith apply PCP to typical scheduling problems such as the Job-Shop Scheduling (JSSP) and the Hoist Scheduling Problem with very encouraging results. Cheng and Smith used a formalism based on the PIA in [Meiri 1991] and employed domain-specific heuristics. What makes DTP and PCP solving similar, and distinguishes them from most other automated scheduling techniques, is their use of the meta-CSP approach. This contrasts with scheduling algorithms that formulate the problem as a CSP with variables that are the start times of the events, and directly solve that CSP.

Stergiou and Koubarakis [Stergiou and Koubarakis 2000] applied their DTP solver on JSSP with somewhat disappointing results. However, it bears remembering that they were comparing a fairly general-purpose temporal reasoning module (their DTP solver) against many highly optimized algorithms that had been tuned specifically for job-shop scheduling problems. It is also worth noting that, JSSP problems are typically optimization problems: it is often relatively easy to find a solution, but very hard to find an optimal solution. In contrast, at least on the random DTP problems we have tested, just finding one solution is inherently hard.

8. Discussion, Contributions, and Future Work

In this paper, we have focused on the development of efficient techniques for solving Disjunctive Temporal Problems (DTPs). DTPs are a class of constraint-based temporal reasoning problems that appeared in the literature for the first time only in 1998 [Stergiou and Koubarakis 1998]. Although DTPs are potentially very useful for a range of planning and scheduling problems, solving them can be computationally quite costly. We therefore examined the strategies that had been proposed in the previous literature for improving the efficiency of DTP-solving, considered how to integrate these strategies with one another and with no-good learning, and conducted systematic experiments to determine what combination of strategies is most effective.

Our experiments were conducted using a DTP-solving system that we implemented, Epilitis. Epilitis is instrumented so that the user can "turn on" various pruning strategies. It is publicly available[20], and may be used as a testbed for further exploration of DTP solving. In our own experiments, we were able to achieve a speed-up of almost two orders-of-magnitude over the previous fastest algorithm, by combining a set of pruning strategies, adding no-good learning with an experimentally determined size bound, and using a carefully analyzed search heuristic.

One important result of our experiments was the discovery that no-good learning is particularly powerful in improving the efficiency of DTP-solving. We can explain this by noting

[20] Contact the first author at ioannis.tsamardinos@vanderbilt.edu. Epilitis will also be available on the first author's web site in the future.

that, in a DTP solver, forward-checking a value requires the propagation of the corresponding STP-constraint in the current STP, which is a relatively costly (albeit polynomial) operation. Thus, even though there is computational overhead associated with retrieving no-goods, this overhead may be outweighed by the savings in forward-checking. In an ordinary (non-temporal) CSP, forward-checking may be less expensive, and the benefits of no-good recording might not be as great.

We have already demonstrated the practical usefulness of DTP-solving in general, and Epilitis in particular, in two of our other research projects. In one of these, the Plan Management Agent (PMA) [Pollack and Horty 1999], we are designing an intelligent calendar that manages a user's plan. In the other, the Autominder [Tsamardinos, Vidal et al. 2002], we are designing a cognitive orthotic system intended to manage and monitor the daily activities of an elderly user, providing him or her with appropriate, timely reminders. Epilitis plays a central role in both systems, serving as the main engine for updating and modifying the modeled plans.

There are many avenues for future work on this topic, and here we mention just a few.

Explore dynamic DTPs. Dynamic DTPs are sequences of DTPs that differ from successive elements by a few constraints. Such sequences arise naturally in certain planning problems in which new goals are periodically added to an existing set of goals. No-good recording may be especially useful for dynamic DTPs, as it is possible that some of the no-goods identified and recorded for the previous DTP will still hold for the next DTP in the sequence, and thus can prune the search space. In one extreme case, if the no-good $<\neg, J>$ still holds in the next DTP, then the DTP is inconsistent and this is proven without any search performed! In [Schiex and Verfaillie 1994] it is shown that the method greatly improves the performance of CSP solvers on dynamic (non-temporal) CSPs.

Consider replacing forward-checking as the underlying algorithm in Epilitis with a full looking ahead approach, investigating the interactions between such an approach and the various speed-up techniques discussed in this paper.

Investigate the use of random restart techniques [Gomez, Selman et al. 1998] to supplement the efficiency-increasing strategies already described in this paper. It seems plausible that there will be a synergistic influence between no-good recording and random restarts. We also expect that certain scheduling techniques, such as profiling, could be applied to DTP solving.

Use DTPs to model conditional plans. As mentioned in Section 7.1, we noted that the *n*-ary constraints of DTPs can be used to model plans with conditional (if-then) branches. In work already underway, we have analyzed the notion of consistency in conditional temporal networks, and have shown that consistency checking in these networks can be reduced to DTP-solving [Tsamardinos, Vidal et al. 2002].

Introduce temporal uncertainty. Even though DTPs are very expressive, subsuming a large number of other temporal and scheduling problems, they have one key limitation:

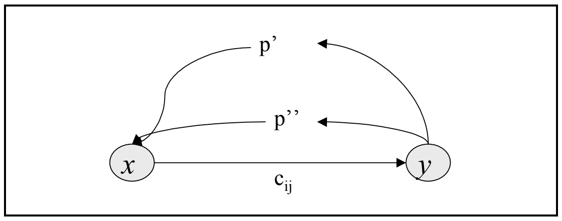


Figure 16: Proof of Theorem 1

they do not readily permit one to model events whose time of occurrence is not known and is not under the control of a planning agent. The recently developed STPU formalism does support modeling of such events, but only as extensions to STPs. It would be very useful to develop techniques for combining STPU and DTPs.

Appendix A: Proofs of Theorems

Let us denote by $d_{xy}(S)$ the distance between time-points x and y in STP S and by $d_{xy}(A)$ the distance between time-points x and y in the STP induced by assignment A.

Theorem 1: A value $cij : y - x = bxy$ is inconsistent with a consistent STP S (that is, S . cij is inconsistent) if and only if the following condition holds:

$$b_{xy} + d_{yx}(S) < 0 \text{ (\textbf{FC-condition})}$$

Proof: Let p_{yx} be the shortest-path from y to x in S and thus the length of p_{yx} is d_{yx}. If the FC-condition holds, then the path p_{yx} . (x,y) has length $b_{xy} + d_{yx}(S)$ and so it forms a negative cycle making S inconsistent.[21]

Conversely, let us suppose that the FC-condition is false and prove that $S' = S$. c_{ij} will be consistent. We will prove this claim by contradiction, i.e. we will assume FC condition is false and S' is inconsistent, and will derive a contradiction. If S' is inconsistent then there is a negative cycle, and because S was consistent before we added c_{ij}, that means that the negative cycle involves the new constraint c_{ij}. Let us assume the negative cycle is c_{ij} . p'_{yx}, for some path p'_{yx}. So $b_{xy} + d''_{yx} = b_{xy} + length(p'_{yx}) < 0$ (1), where d''_{yx} is the distance between y and x in S' for some path p'', which is the shortest path from y to x (Figure 16). Since the negative cycle is a simple cycle (no loops allowed), then edge (x, y) is not a member of p''. Therefore, the shortest path p'' from y to x in S' is does not contain the new constraint c_{ij} and thus distance from y to x in S' and S is the same: $d_{yx} = d''_{yx}$. By (1) above we get that $b_{xy} + d''_{yx} = b_{xy} + d_{yx} <0$, i.e. FC-condition holds, contrary to what we assumed.

Theorem 2: A value $c_{ij} : y - x = b$ is subsumed by an STP S if and only if $d_{xy}(S) = b$ (***Subsumption-Condition***), where $d_{xy}(S)$ is the distance between x and y in S.

[21] Let p_{ab} be a path $(a, n_1, \ldots, n_k,b)$ in a graph G, and let c be a node in G. Then p_{ab} . c is the path $(a, n_1, \ldots, n_k,b,c)$ in G.

Proof: “. ” Suppose that the Subsumption-Condition holds for value c_{ij} . By definition of the distance $y - x = d_{xy}(S)$ holds in all exact STP solutions, so $y - x = d_{xy}(S) = b$ holds in all exact solutions, and c_{ij} holds in all exact solutions of S. “⇒” Conversely, suppose the Subsumption-Condition does not hold, i.e. $b < d_{xy}(S)$ holds. Then there is some exact solution s of the STP for which $y - x = d_{xy}(S)$ (as shown in [Dechter, Meiri et al. 1991]). Thus, in s, $b < y - x$, i.e. c_{ij} does not hold in all of S’s exact solutions.

Lemma 1: Adding more constraints to an STP can only result in monotonic decrease in the distances between nodes.

Proof: Immediate, by the fact that adding a constraint to an STP can only reduce the solution set, and thus the distances have to be smaller or equal to the original STP.

Lemma 2: Adding a value $c{:}\{y - x = b\}$ to an STP S when S subsumes c, does not change the distance array of S (i.e. the resulting STP is equivalent to the one before the addition).

Proof: Since c is subsumed, by Theorem 2, $d_{xy}(S) = d$. Let p be a shortest path in the distance array from x to y, and so its weight is $d_{xy}(S)$. Let us suppose now that the distance array of S changes when c is added. That means that the new constraint c participates in at least one shortest path, let us say on the path from w to z that before the addition had distance $d_{wz}(S)$. From shortest path properties we get:

$$d_{wz}(S) = d_{wx}(S) + d_{xy}(S) + d_{yz}(S) = d_{wx}(S) + b + d_{yz}(S) \quad (1)$$

After the addition, in the new STP $S' = S \; . \; c$, the new distance $d_{wz}(S')$ is:

$$d_{wz}(S') = d_{wx}(S') + b + d_{yz}(S') = d_{wx}(S) + b + d_{yz}(S) \quad (2)$$

Notice that $d_{wx}(S') = d_{wx}(S)$ and $d_{yz}(S') = d_{yz}(S)$ because b is already participating in the shortest path from w to z and therefore cannot participate on the shortest paths from w to y or from x to z or a cycle would be present on the path from w to z (i.e. the shortest paths would not be simple).

Since the distance array changed, the new shortest path has to be strictly smaller the one than before c was added (Lemma 1), i.e.

$$d_{wz}(S') < d_{wz}(S) \quad (3)$$

(1) and (2) imply that $d_{wz}(S) = d_{wz}(S')$ which contradicts (3). Therefore our initial assumption that the distance array will change is false.

Theorem 3: Let $D=<V, C>$ be a DTP, let A be an assignment on D (i.e. a component STP), and let C_i be a variable subsumed by A. Then A is a solution of D if and only if it is a solution of $D'=<V, C - C_i>$.

Proof: Since we assume that C_i is a subsumed variable, then, there must be a value $c : y - x = b$ in the domain of C that is subsumed by A. Suppose that A is a solution of D. Then obviously, it is a solution of $D' = <V, C - C_i>$ since D' has one less variable (DTP constraint). Conversely, suppose A is a solution of D'. Since c is subsumed by A, it holds in all of A's exact solutions, and thus C_i which is a disjunction involving c, holds in all of A's exact solutions. Thus, if A is a solution of the DTP constraint $C - C_i$, it also solves the DTP constraints C.

Corollary 1: Let A be a partial assignment during a DTP search, U be the unassigned variables, and *Sub* be the set of subsumed variables in U. If A can be extended to a solution over variables in U – *Sub*, it can be extended to a solution over variables in U. In other words, we can remove the subsumed variables from the unassigned variables during search. The solution to the reduced problem is a solution to the original

Proof: By Theorem 3 if A' is an extension of A over the variables at U – Sub, and A' is consistent, then A' is also a solution to the original DTP.

```
1. Epilitis(A, U, S) /* A is the set of assigned CTP variables, U the set of unassigned variables, and S a distance and
predecessor array representing the current STP */
2*      If U=¬ Then
3*          A is a solution, Stop
4*      Else
5*          Let x be a variable in U, J =¬, BJ = false
6. SB       SBJ =¬                      /* Set the semantic branching justification to empty */
7. RSV      If there is value v.  d(x) subsumed by S Then
8. RSV              Return Epilitis(A , U – {x }, S)
9. RSV      EndIf
10*         For each v .  d(x) until BJ    /* loop for all values in the current domain or until the BJ flag is true */
11*                 A' = A .  {x .   v } /* add value to a (new) assignment */
12.                 S' = maintain-consistency(v, S)
13.                 If S' is inconsistent Then
14. SB                      SBJ = justification-value(v, S')
15.                         J = J .  SBJ
16.                         GoTo 36
17.                 EndIf
18. FC-off          If d(x) is singleton  /* when FC-off omit forward checking when d(x) is singleton */
19. FC-off                  K = ¬
20. FC-off          Else
21*                         K be forward-check(A', U, S')
22.                 EndIf
23*                 If K = ¬  /* If we have not reach a dead end … */
24*                         Let J-sons be Epilitis(A' , U – {x}, S' ) /* then recursively call Epilitis */
25*CDB                      If x .  J-sons Then
26*                                 J .   J .   J-sons
27*CDB                      Else  /* If the current variable does not participate in the failure justification */
28*CDB                              J .   J-sons, BJ=true          /* then exit the loop */
29* CDB                     Endif
30. SB                      SBJ =J-sons
31*                 Else
32*                         J .   J .   K
33*NG                       record(project(A', K), K)
34. SB                      SBJ = K
35*                 Endif
36. SB              If v is not the last value in d(x)  /* remember SB-constraints is the only global variable */
37. SB                      S=maintain-consistency(S, reverse(v));  SB-Constraints = SB-constraints .   <re-
verse(v), SBJ>
38. SB                      If S is inconsistent
39. SB                              un-forward(x), FinishLoop
40. SB                      EndIf
41. SB              EndIf
42*                 un-forward(x)
43*         EndFor
44*         If BJ = false Then
45*                 For each v .  D(v) –  d(v)  /* add the justifications that removed the values */
                                                /*  from the current domain */
46*                         J .   J .   killers(v)
47*                 EndFor
48*NG               record(project(A, J), J)
49*         Endif
50. SB      Remove all semantic branching constraints added in this invocation of Epilitis from SB-Constraints
51*         Return J
52*     Endif
```

Figure 17: The Epilitis Algorithm

Appendix B: The Epilitis Algorithm in Detail

Epilitis, shown in Figure 17, is a generalization of the no-good recording algorithm in [Schiex and Verfaillie 1994] (see Figure 4 *ibid*), and it just adds code to this algorithm. *The lines common to both algorithms are annotated with an asterisk* following the line number[22]. Epilitis would still correctly solve DTP problems if only these lines are included. The only difference from the plain no-good recording algorithm would be in forward checking. Epilitis' forward checking mechanism is similar to the one in Figure 2, which takes into consideration the fact that the values of the meta-CSP express STP-like constraints. In other words, the algorithm in [Schiex and Verfaillie 1994] (Figure 4), with a modified forward-checking function, is a no-good recording DTP solver.

Having said that, two points must be explained regarding the workings of Epilitis: (i) the additional ("un-starred") lines in the algorithm and (2) the exact way forward checking is performed. The former is the topic of the rest of this section, and the latter the topic of the next one.

For the rest of the discussion let us assume that there is available a function **forward-check**(*A, U, S*) that given the assignment *A*, removes from the variables *U* all the values in their current domains that are inconsistent with *A*. To check the FC-Condition efficiently we provide to the function the distance array *S* that corresponds to *A*. If a domain of a variable is reduced to the empty set, then **forward-check** should return a justification *K* (also called the *value Killers* of the domain values; see also [Tsamardinos 2001] (Section 3.5) and [Schiex and Verfaillie 1994]), which is a minimal set of variables in *A* such that the constraints among them cause all the variables of the domain to be removed.

The annotations **SB, FC-off, CDB, NG** and **RSV** shown next to a line in the algorithm indicate which of the corresponding techniques (semantic branching, FC-off, conflict directed backjumping, no-good recording, and removal of subsumed variables, respectively) the line serves. For example, to remove semantic branching from the algorithm, we could just remove the lines annotated with **SB**.

The ***Removal of Subsumed Variables*** (RSV) in lines 7-9 is achieved by testing if, in the next variable to assign, there is a value that is subsumed by the current STP *S*. The test can be achieved by checking the Subsumption-Condition of Theorem 2. If the variable is subsumed, then it is removed from the unassigned variables and Epilitis is recursively called.

Line 12 propagates the value/constraint of assignment {*x* .   *v*} in the current STP *S'* so that the distances of the STP corresponding to the current assignment *A'* are available with a simple table lookup. Recall that the distances are required to calculate both the FC-Condition and the Subsumption-Condition. This technique of maintaining the distance array was described in full in Section 2.2 and presented in the algorithm in Figure 2.

[22] The additional lines 45-47 are not in the original paper [Schiex and Verfaillie 1994]. In direct communication with the first author of the paper it was established that lines 45-47 are indeed required for the algorithm to be complete. The experimental results in that paper are not invalidated however because lines 45-47 were included in the implementation and were only missing from the pseudo-code description of the algorithm. Careful implementation of the Epilitis algorithm also revealed a typo in the original publication of the algorithm. Line 33 appears originally as record(project(A, K), K) while it should be record(project(A', K), K).

When only the basic forward checking DTP solving algorithm is used, no assignment {*x* . *v* } will ever cause an inconsistency to the current STP because, if it did it would have been removed by forward checking. However, when semantic branching is used, it becomes necessary to check that the constraint {*x* . *v* } added by semantic branching does not cause an inconsistency. This is the reason for the check at line 13. If we have indeed hit an inconsistency the reason for it is accumulated in variable *J* (line 15), which is the justification to return in case of a failure (line 50). Line 14, annotated with **SB,** is explained in the discussion of semantic branching below.

Next the algorithm performs FC-off (lines 18-20). Very simply, if there is only one value in the current domain of the current variable, we omit forward checking and assume that it succeeded by setting *K* = ¬. The ramifications of the omission were the subject of Section 6.3 above. Otherwise, we perform forward checking and store in *K* the *value killers* of the domain that was reduced to the empty set or we store ¬ to *K* if there is no such domain.

If we have not hit a dead-end, i.e. *K*=¬ (line 23), then we recursively call Epilitis. If it returns, then we have failed to extend the current assignment *A'* to a solution and the return value is a justification of the failure, stored in the variable *J-sons.* If the current variable participates in this justification (line 25) then we accumulate the justifications in variable *J* and proceed (after line 36) trying new values for the current variable. If on the other hand, the current variable has nothing to do with the failure, we jump to line 28, where *BJ* (from backjumping) is set to *true* so that we exit the loop and avoid trying any other values of the current variable, and finally return the same reason *J-sons* that caused the failure in the recursive call. On the other hand, if forward check fails, it returns the value killers *K* that are accumulated in the overall justification *J* (line 32). Line 33 records the no-good implied by the dead-end. Lines 23-35 (apart from the addition of lines 30 and 34) are exactly the same as in the non-temporal no-good recording algorithm.

Perhaps the most complicated addition is the lines that achieve semantic branching. Integrating SB with the rest of the pruning techniques has implications that also affect the details of forward checking but for the moment we restrict the discussion only to the code that appears in Figure 17. When the current assignment *A* . {*x* . *v* } fails to extend to a solution, the code reaches line 36. As already described in detail in Section 3.3, for the rest of the search under assignment *A*, we can assume that ↔*v* does not hold. Thus, line 36 propagates the **reverse** of *v* in the current STP *S* (i.e. the STP that corresponds to assignment *A*). The propagation might cause an inconsistency which would be identified at line 38 in which case there is no reason to try a different value for variable *x* and we can exit the loop.

For reasons that we explained in Section 5, it is necessary to store the semantic branching constraints that we propagate along with the justification for their addition. The store occurs at line 37 where pairs <*v, SBJ*> are stored in the global variable *SB-Constraints*, where *v* is an STP-like constraint and *SBJ* a justification (from semantic branching justification). There are three different reasons why the current value *v* causes an inconsistency, and correspondingly, three different lines where *SBJ* is assigned a value. Value *v* might directly cause an inconsistency in the current assignment *A* in which case *SBJ* is the justification discovered by function **justifi-**

cation-value (Figure 4)[23] and the assignment takes place at line 14. Alternatively, if after assigning value *v* forward check failed, then *SBJ* should be the value killers *K* that forward check returned (line 34). Finally, if after assigning value *v* forward checked succeeded but the recursive call to Epilitis failed, then *SBJ* is assigned the value *J-sons* (line 30). In all three cases, we fail to extend *A* . {*x* . *v* } to a solution and we store in *SBJ* the reason for the failure (i.e. the culprit of the variables participating in the constraints that cause the inconsistency).

The rest of the Epilitis algorithm, as already mentioned, is exactly the same as the non-temporal no-good recording algorithm in [Schiex and Verfaillie 1994].

Forward Checking and Justifications in Epilitis

Figure 17 presented the Epilitis algorithm, however, important details for its implementation were hidden in the **forward-check** and **justification-value** functions. We now proceed to the discussion of these two functions.

Recall that we assumed that **forward-check**(*A, U, S*) is a function that, given the assignment *A*, removes from the variables *U* all the values in their current domains that are inconsistent with *A*. The STP *S* containing the distance array that corresponds to *A* is passed to efficiently check the FC-Condition. A very important feature of **forward-check** is that if a domain of a variable is reduced to the empty set, it should return a justification *K* (also called the *value killers*), which is a minimal set of variables in *A* whose constraints cause the variables of the domain to be removed.

Forward-check should check if each remaining value *v* in some current domain of a variable should be removed or not. A value *v* should be removed, if, as before, the FC-Condition holds; is should also be removed if *A* . {*x* . *v* } is a superset of some recorded no-good <*A', J*>, i.e. if *A'* . *A* . {*x* . *v* }. That achieves forward checking, but it does not solve the problem of assembling and returning a justification in the case where a variable domain is reduced to the empty set. Let us suppose that **justification-value**(*v, S*) is responsible for returning the justification of the removal of a single value *v* given the current STP *S*. Then, the overall justification for a variable domain being empty is the union of the justifications for removing each value originally in that domain.

[23] Function **justification-value** has to be slightly modified from Figure 4 to work for Epilitis as we will see in the next section.

```
forward-check(A, S, U)
1.  For each variable C in U
2.      For each value $c : x - y = b_{xy}$ in $d(C)$
3.      If $b_{xy}$ + distance (y, x, S) < 0 (FC-Condition) or
4.        A .   {C .   c } is a superset of A', where <A', no-good-J> is a recorded no-good
5.          Remove c from d(C)
6.          If d(C) = ¬
7.              K = ¬
8.              For each value v in D(C)
9.                  K = K .   justification-value(v, A, S)
10.             EndFor
11.             return K
12.         EndIf
13.     EndIf
14. EndFor
15. Return ¬
```

Figure 18: Forward Checking for Epilitis

```
justification-value(c : y – x = b , A, S}
1. If A .   {C .   c } is a superset of A', where <A', J'> is a recorded no-good
2.     Return J'
3. Else
4.     p = shotest-path( y, x, S)
5.     Return vars(p .   c} .   {J, where <v, J> .   SB-Constraints and v .   p}
6. EndIf
```

Figure 19: The function just-value for Epilitis

Now we can turn our attention to the implementation of the function **justification-value**(*v, A, S*) which should return a set of variables from *A*, i.e. a justification that explains why *A* .   {*C* .   *v* } cannot be extended to a solution and thus why *v* has to be removed from the current domain of *C*. Trivially, all the variables in *A* plus the variable *C* constitute a justification for the removal of *v*. However, we can find smaller justifications that provide more opportunities for conflict directed backjumping and search pruning.

There are two reasons why a value *v* might be removed. The first one is if *A* .   {*C* .   *v*} is a superset of *A'*, where <*A'*, *J'*> is a recorded no-good, and then the justification for removing *v* is *J'*[24]. The second reason is if *v* is removed because *A* .   {*C* .   *v*} corresponds to an inconsistent STP *S*. Then, as explained in detail in Section 4, the variables that cause the inconsistency are the ones that have values assigned to them that participate in a negative cycle in *S*.

As already mentioned in Section 5, when semantic branching is present, the current STP *S* does not directly correspond to the current assignment *A*, but instead is formed by all the constraints in *A* plus the semantic branching constraints added. Thus, in Epilitis, when the negative cycles are identified, the corresponding justification is not just the variables with values that participate in the cycle, but also the variables (i.e. the justifications) that were responsible for the addition of the semantic branching constraints that participate in the cycle. These justi-

[24] See [Tsamardinos 2001] for a complete description of the algorithm for storing and retrieving no-goods used in Epilitis.

fications can be found in the *SB-Constraints* structure: whenever a semantic branching constraint is propagated the pair <*←v, SBJ*> is stored at *SB-Constraints* (line 37, Figure 17).

Appendix C: FC-Off Reduces the Number of Forward-Checks but Increases Solving Time

With FC-off, when the current domain $d(C_i)$ of a variable C_i has been reduced to a singleton set $\{c_{ij}\}$, forward checking is suspended and the constraint c_{ij} is assigned to C_i without forward checking. FC-off is illustrated in the following example:

Example 6: Consider the following DTP:

$$
\begin{aligned}
&C_1 : \{c_{11} : y - x = 5\}\\
&C_2 : \{c_{21} : x - z = 5\}\\
&C_3 : \{c_{31} : v - x = 5\} \ . \ \{c_{32} : z - v = 10\}\\
&C_4 : \{c_{41} : y - z = -10\} \ . \ \{c_{42} : x - y = -10\}
\end{aligned}
$$

Without FC-off, when the current assignment becomes $A_1 = \{C_1 \ . \ c_{11}\}$, forward checking will remove variable c_{42} from $d(C_4)$. In the next step, when the current assignment is $A_2 = \{C_1 \ . \ c_{11}, C_2 \ . \ c_{21}\}$, c_{41} is also removed and thus $d(C_4)$ becomes empty and the search returns failure. In contrast, when FC-off is used, when the current assignment is $A_1 = \{C_1 \ . \ c_{11}\}$ forward checking is suspended and nothing is removed from any variable's domain. Similarly, when the current assignment becomes $A_2 = \{C_1 \ . \ c_{11}, C_2 \ . \ c_{21}\}$, forward checking is still turned off, and so still nothing is removed from any variable's domain. Only when the non-singleton domain $d(C_3)$ is encountered, and the current assignment becomes $A_3 = \{C_1 \ . \ c_{11}, C_2 \ . \ c_{21}, C_3 \ . \ c_{31}\}$ is forward check called; at this point, it will recognize the failure. Notice that when FC-off is not used, the algorithm forward checks a total of 5+3=8 values (i.e., it performs five checks for A_1—one for each of the other values—removing one of those values; and three checks for A_2). However, it only expands two nodes. In contrast, with *FC-off*, the algorithm checks 2 values (performed only when the current assignment is A_3) and it expanded three nodes.

The above example shows that a technique such as FC-off may or may not increase the performance of a DTP solving algorithm. With FC-off there is less forward checking but more nodes are expanded (Theorem 17 in [Stergiou and Koubarakis 2000]). Therefore, overall effect of the FC-off technique will depend on the relative time required to expand nodes and to perform forward checking. There has not been a theoretical analysis of the conditions under which FC-off improves performance, but our experiments, shown below, suggest that *FC-off* frequently degrades performance even though it reduces the number of forward checks[25].

[25] In contrast, Stergiou and Koubarakis note "We have also measured the CPU times used by the algorithms we studied. As expected, the CPU times are proportional to the number of consistency checks," ([Stergiou and Koubarakis 2000], Sec. 6). We hypothesize that this is because in their implementation each consistency-check was not a simple array lookup. Instead, it involved one constraint propagation and one constraint retraction. Thus, the total time spent in forward-check greatly dominates the time spent in maintain-consistency.

The arguments just given about the flaws in using *CC* counts as a metric of performance are supported by our experiments, as illustrated in Table 8 and Table 9, which show how well different combinations of pruning strategies worked in Epilitis. First consider Table 8, which shows the results for N=30. The first column orders the algorithms in decreasing order according to the median time taken in the critical region where R=6. The second column orders the algorithms by using the median number of *CCs,* from highest to lowest. (So, in both columns, combinations that are "better" are listed at the bottom on the column.) It is easy to see that the median *CCs* favors the algorithms that use FC-off (denoted with an FC in their name) and ranks them the best while in fact they are the worst in terms of median time performance.

N=30	
Median-Time	**Median-CCs**
SB-FC	SB
CDB-SB-RSV-FC	CDB-SB
SB	SB-RSV
CDB-SB-RSV	CDB-SB-RSV
CDB-SB	SB-FC
SB-RSV	CDB-SB-RSV-FC

Table 8: The ordering of performance for N=30, R=6, from worst (top) to best performance.

N=20	
Median-Time	**Median-CCs**
Nothing	Nothing
CDB-RSV-FC	RSV
CDB-FC	CDB
RSV	CDB-RSV
CDB	SB
CDB-RSV	SB-RSV
CDB-SB-RSV-FC	CDB-SB
SB-FC	CDB-SB-RSV
SB	CDB-FC
SB-RSV	CDB-RSV-FC
CDB-SB	SB-FC
CDB-SB-RSV	CDB-SB-RSV-FC

Table 9: The ordering of performance for N=20, R=6, from worst (top) to best performance.

We repeated the same procedure as above for N=20, for which a greater number of experiments of pruning combinations were available. The results are displayed in Table 9. Note again that there is a large disparity between the ranking by *CC* count and the ranking by time. For instance, using the *CC* metric, CDB-SB-RSV-FC is ranked the best, five positions higher than it really is when we consider execution time.

References

[Armando, Castellini et al. 1999] Armando, A., C. Castellini and E. Giunchiglia (1999). SAT-based Procedures for Temporal Reasoning. 5th European Conference on Planning (ECP-99).

[Barber 2000] Barber, F. (2000). "Reasoning on Interval and Point-based Disjunctive Metric Constraints in Temporal Contexts." Journal of Artificial Intelligence Research **12**: 35-86.

[Bessiere 1999] Bessiere, C. (1999). Non-binary constraints. Principles and Practice of Constraint Programming (CP'99), Alexandria, Virginia, USA, Springer.

[Bessiere, Mesequer et al. 1999] Bessiere, C., P. Mesequer, J. Larrosa and E. Freuder (1999). On forward-checking for non-binary constraint satisfaction. CP'99, Alexandria, VA.

[Bessiere and Regin 1997] Bessiere, C. and J. C. Regin (1997). Arc consistency for general constraint networks: preliminary results. International Joint Conference on Artificial Ingelligence (IJCAI'97(), Nagoya, Japan.

[Chen and Beek 2001] Chen, X. and P. v. Beek (2001). "Conflict-Directed Backjumping Revisited." Journal of Artificial Intelligence Research **14**: 53-81.

[Cheng and Smith 1995] Cheng, C. and S. F. Smith (1995). Applying constraint satisfaction techniques to Job-Shop Scheduling.Technical Report CMU-RI-TR-95-03 Pittsburgh, Carnegie Mellon University.

[Cheng and Smith 1995] Cheng, C. and S. F. Smith (1995). A constraint posting framework for scheduling under complex constraints. Joint IEEE/INRIA conference on Emerging Technologies for Factory Automation, Paris, France.

[Chleq 1995] Chleq, N. (1995). Efficient Algorithms for Networks of Quantitative Temporal Constraints. Constraints'95.

[Cormen, Leiserson et al. 1990] Cormen, T. H., C. E. Leiserson and R. L. Rivest (1990). Introduction to Algorithms. Cambridge, MA, MIT Press.

[Dechter 1990] Dechter, R. (1990). "Enhancement schemes for constraint processing: backjumping, learning, and cutset decomposition." Artificial Intellience **41**.

[Dechter and Frost 1999] Dechter, R. and D. Frost (1999). Backtracking algorithms for constraint satisfaction problems.Technical Report, University of California at Irvine.

[Dechter, Meiri et al. 1991] Dechter, R., I. Meiri and J. Pearl (1991). "Temporal constraint networks." Artificial Intelligence **49**: 61-95.

[Frost and Dechter 1994] Frost, D. and R. Dechter (1994). Dead-end driven learning. National Conference in Artificial Intelligence (AAAI-94).

[Ginsberg 1993] Ginsberg, M. (1993). "Dynamic Backtracking." Journal of Artificial Intelligence Research.

[Ginsberg and McAllester 1994] Ginsberg, M. and D. McAllester (1994). "GSAT and Dynamic Backtracking." Journal of Artificial Intelligence Research.

[Gomez, Selman et al. 1998] Gomez, C. P., B. Selman and H. Kautz (1998). Boosting combinatorial search through randomization. National Conference on Artificial Intelligence (AAAI'98), Madison, Winsonsin.

[Meiri 1991] Meiri, I. (1991). Combining qualitative and quantitative constraints in temporal reasoning. National Conference in Artificial Intelligence (AAAI'91).

[Mohr and Henderson 1986] Mohr, R. and T. C. Henderson (1986). "Arc-consistency and path-consistency revisited." Artificial Intelligence **28**(225-233).

[Oddi and Cesta 2000] Oddi, A. and A. Cesta (2000). Incremental Forward Checking for the Disjunctive Temporal Problem. European Conference on Artificial Intelligence.

[Pollack 2002] Pollack, M. E. (2002). Planning Technology for Intelligent Cognitive Orthotics. 6th International Conference on AI Planning and Scheduling.

[Pollack and Horty 1999] Pollack, M. E. and J. F. Horty (1999). "There's More to Life than Making Plans: Plan Management in Dynamic   Environments." AI Magazine.

[Prosser 1993] Prosser, P. (1993). "Hybrid algorithms for the constraint satisfaction problem." Computational Intelligence **9**.

[Richards 1998] Richards, E. T. (1998). Non-systematic Search and No-good Learning.Ph.D. Thesis IC Parc. London, Imperial College.

[Roberto J. Bayardo and Schrag 1977] Roberto J. Bayardo and R. C. Schrag (1977). Using CSP Look-Back Techniques to Solve Real-World SAT Instances. Fourtheeen National Conference on Artificial Intelligence (AAAI'97).

[Schiex and Verfaillie 1994] Schiex, T. and G. Verfaillie (1994). "Nogood Recording for Static and Dynamic Constraint Satisfaction Problems." International Journal of Artificial Intelligence Tools **3**(2): 187 - 200.

[Schiex and Verfaillie 1994] Schiex, T. and G. Verfaillie (1994). Stubbornness: a possible enhancement for backjumping and no-good recording. European Conference in Artificial Intelligence (ECAI-94).

[Staab 1998] Staab, S. (1998). On Non-Binary Temporal Relations. European Conference on Artificial Intelligence.

[Stergiou and Koubarakis 1998] Stergiou, K. and M. Koubarakis (1998). Backtracking Algorithms for Disjunctions of Temporal Constraints. 15th National Conference on Artificial Intelligence (AAAI-98).

[Stergiou and Koubarakis 2000] Stergiou, K. and M. Koubarakis (2000). "Backtracking algorithms for disjunctions of temporal constaints." Artificial Intelligence **120**(1): 81-117.

[Tsamardinos 1998] Tsamardinos, I. (1998). Reformulating Temporal Plans for Efficient Execution.Masters Thesis. Pittsburgh, University of Pittsburgh.

[Tsamardinos 2001] Tsamardinos, I. (2001). Constraint-Based Temporal Reasoning Algorithms with Applications to Planning.Ph.D Thesis Computer Science Department. Pittsburgh, University of Pittsburgh.

[Tsamardinos, Vidal et al. 2002] Tsamardinos, I., T. Vidal and M. E. Pollack (2002). "CTP: A New Constraint-Based Formalism for Conditional, Temporal Planning." Constraints (to appear).

[Yokoo 1994] Yokoo, M. (1994). Weak-commitment search for solving constraint satisfaction problems. National Conference in Artificial Intelligence (AAAI-94).

APPENDIX B

CTP: A New Constraint-Based Formalism for Conditional, Temporal Planning

Ioannis Tsamardinos (ioannis.tsamardinos@vanderbilt.edu)
Department of Biomedical Informatics, Vanderbilt University

Thierry Vidal (thierry@enit.fr)
Production Engineering Laboratory (LGP) - ENIT, France

Martha E. Pollack (pollackm@eecs.umich.edu)
Computer Science and Engineering, University of Michigan

Abstract. Temporal constraints pose a challenge for conditional planning, because it is necessary for a conditional planner to determine whether a candidate plan will satisfy the specified temporal constraints. This can be difficult, because temporal assignments that satisfy the constraints associated with one conditional branch may fail to satisfy the constraints along a different branch. In this paper we address this challenge by developing the Conditional Temporal Problem (CTP) formalism, an extension of standard temporal constraint-satisfaction processing models used in non-conditional temporal planning. Specifically, we augment temporal CSP frameworks by (1) adding observation nodes, and (2) attaching labels to all nodes to indicate the situation(s) in which each will be executed. Our extended framework allows for the construction of conditional plans that are guaranteed to satisfy complex temporal constraints. Importantly, this can be achieved even while allowing for decisions about the precise timing of actions to be postponed until execution time, thereby adding flexibility and making it possible to dynamically adapt the plan in response to the observations made during execution. We also show that, even for plans without explicit quantitative temporal constraints, our approach fixes a problem in the earlier approaches to conditional planning, which resulted in their being incomplete.

1. Introduction

Classical planning (Smith et al., 2000) assumes that a plan can be generated off-line, prior to its execution, and that execution consists in the straightforward activation of the steps in the plan, in an order that is consistent with the plan's temporal constraints. This assumption is satisfied in domains in which the planning agent is omniscient, and thus knows at plan time everything required about the possible evolution of the world during execution. In many real-world domains, this assumption is violated.

One way to handle this difficulty is to build the plan on-line, making all decisions in a reactive fashion. However, the reactive approach has a number of shortcomings; in particular, when there are real-time requirements to be satisfied, a reactive approach typically cannot guarantee that they will be met (nor can it determine that they are unsatisfiable). In addition, a reactive approach will fail to take preparatory steps that might be required for the continuation of the plan in unexpected circumstances. Consequently, an alternative approach has been to develop conditional planning capabilities (Peot and Smith, 1992; Pryor and Collins, 1996; Onder and Pollack, 1999). In the conditional planning approach, plans are generated prior to execution, but they include both observation actions and conditional branches, which may or may not be executed, depending on the execution-time result of certain observations.

In addition to omniscience, classical planning also assumes instantaneous, atomic actions. However, modern planning systems (Muscettola et al., 1998; Laborie and Ghallab, 1995) indicate the growing need to represent and reason with metric temporal information and constraints. Temporal constraints pose a challenge to conditional planning. In particular, it is necessary for a temporal conditional planner to determine whether a candidate plan can possibly satisfy the specified temporal constraints. This can be difficult, because temporal assignments that satisfy the constraints associated with one conditional branch may fail to satisfy the constraints along a different branch.

In this paper we address this challenge by developing the **Conditional Temporal Problem** (CTP) formalism, an extension of standard temporal constraint-satisfaction processing models used in non-conditional temporal planning. Specifically, we augment the previous models by (1) adding observation nodes, which correspond to the time-points at which observation actions end, and (2) attaching labels to all other nodes in the network. A node's label indicates the situation(s) in which the event it denotes (the start or the end of a plan's action) will be executed.

As we will show, our extended framework allows for the off-line construction of conditional plans that are guaranteed to satisfy complex temporal constraints. Importantly, this can be achieved even while allowing for the decisions about the precise timing of actions to be postponed until execution time, in a least-commitment manner, thereby adding flexibility and making it possible to adapt the plan dynamically, during execution, in response to the observations made. We also show that, even for plans without explicit quantitative temporal constraints, our approach fixes a problem in the earlier approaches to conditional planning, which resulted in their being incomplete.

In Section 2 we review the temporal constraint models that inspired our conditional model, which itself is depicted in Section 3. Three levels of consistency in conditional temporal problems are then defined in Section 4. Each is developed, in turn, in the three subsequent sections, which provide algorithms, discuss theoretical complexity issues, and describe the usefulness of each form of consistency to planning. In particular, in Section 8, we explain how the use of our approach in conditional planning corrects a problem in the earlier approaches. We conclude this article with a discussion of related and future work.

2. Background

A variety of planning systems (Laborie and Ghallab, 1995; Muscettola et al., 1998), often referred to as temporal planners, use an underlying constraint model to query and check the temporal aspects of a plan. One of the most popular models is the *Temporal Constraint Satisfaction Problem* (TCSP) and its graph-based counterpart, the *Temporal Constraint Network* (TCN) (Dechter et al., 1991). A TCN is a constraint graph $< V, E >$ where the nodes V represent time-points (i.e. instantaneous events associated with the start or end of actions), while the edges E represent binary constraints on the duration between two time-points $x_i, x_j \in V$ of the form:

$$(l_1 \leq x_j - x_i \leq u_1) \vee \ldots \vee (l_n \leq x_j - x_i \leq u_n)$$

where $l_1, \ldots, l_n, u_1, \ldots, u_n \in \Re$ are the lower and upper bounds of the constraint. In other words the time from x_i to x_j must lie in one of the intervals $[l_1, u_1]$, $[l_2, u_2]$, $\ldots [l_n, u_n]$. For example, if x and y represent the start and end time-points of action A then the constraint $(5 \leq y - x \leq 10) \vee (20 \leq y - x \leq 25)$ specifies that the duration of A is between 5 and 10 time units (e.g. minutes) or between 20 and 25 time units.

An interesting case is when only one interval $[l, u]$ is allowed for each edge. This is the *Simple Temporal Problem* (STP) restriction, whose graph counterpart is the *Simple Temporal Network* (STN). Checking the consistency of a TCN is known to be NP-complete, while in an STN consistency can be determined in polynomial time, for instance using a local path-consistency propagation algorithm (Mackworth and Freuder, 1985).

On the other hand, TCSPs may be generalized by allowing non-binary constraints. This is the *Disjunctive Temporal Problem* (DTP) (Stergiou and Koubarakis, 2000), which formally is a constraint-satisfaction problem $< V, E >$ such that the constraints E are arbitrary disjunctions of STP-like constraints, i.e. each member of E has the form

$$(l_1 \leq x_{i_1} - x_{j_1} \leq u_1) \vee \ldots \vee (l_n \leq x_{i_k} - x_{j_k} \leq u_n)$$

where again $l_1, \ldots, l_n, u_1, \ldots, u_n \in \Re$. DTPs are thus conjunctions of n-ary disjunctive constraints: $\wedge_i(\vee_j c_{ij})$, where the c_{ij} are of the form $l \leq x - y \leq u$. As such, they can represent any other formula that we can construct with propositions c_{ij} (e.g. $c_{11} \Rightarrow c_{12}$).[1] The most frequently used way to solve a DTP is to convert it to a problem of selecting one disjunct, $(l_k \leq x_i - x_j \leq u_k)$ from each disjunctive constraint, such that the set of selected disjuncts forms a consistent STP. The DTP is consistent if and only if at least one of such *component STP* is consistent. Checking the consistency of a DTP is NP-hard, and because DTPs include non-binary constraints, it is difficult to use path-consistency on them to increase efficiency(Bessiere, 1999). However, several recent papers have presented heuristic techniques that significantly decrease the typical time needed to check DTP consistency (Armando et al., 1999; Stergiou and Koubarakis, 2000; Oddi and Cesta, 2000; Tsamardinos, 2001).

Recently, STPs have also been extended to take into account temporal uncertainty. In STPs (as in TCSPs and DTPs), the constraint between any two time-points x and y specifies an interval that is controllable: I.e. the execution agent can choose for $(y - x)$ any of the values within the allowed bounds given by the constraint. In realistic planning applications however, the durations of some tasks or the time of occurrence of some events may depend on external parameters, and thus the actual value can only be *observed* by the agent at execution time (e.g., (Muscettola et al., 1998)). Such *contingent* values may be seen as being assigned by Nature while the other values are assigned by the executing agent. A *Simple Temporal Problem/Network with Uncertainty* (STPU/STNU) (Vidal and Fargier, 1999) is similar to a STP/STN except that the edges are divided into *contingent* and *requirement* edges. The finishing time-point of contingent intervals are controlled by Nature and hence *observed*, while others are controlled and hence *executed* by the agent. We will refer to this model later in the paper, as there is a strong relationship between consistency in an STNU (called controllability in the STNU setting) and consistency as we define it for CTPs.

3. Conditional Temporal Problems

We now introduce the Conditional Temporal Problem (CTP) formalism. Both in this section and throughout the remainder of the paper, we

[1] Note that this is true despite the fact that DTPs do not include negated literals, because $\neg c_{ij} : l \leq x - y \leq u$ can always be rewritten as $x - y < l \vee x - y > u$, approximated as close as desired by $x - y \leq l - \epsilon \vee x - y \geq u + \epsilon$.

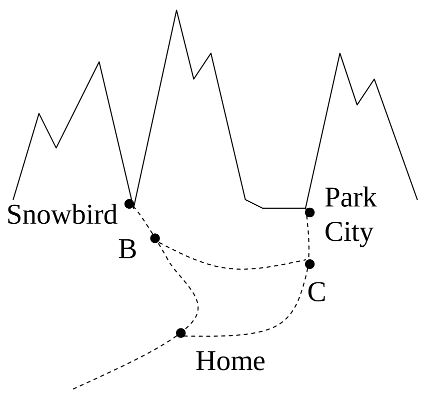


Figure 1. The map for the CNLP plan.

will illustrate our approach with a simple example derived from the one used in the original CNLP conditional-planning paper (Peot and Smith, 1992). The example models a plan for going skiing, either at Snowbird or at Park City, starting from home, and traversing the roads (shown as dashed lines) depicted in Figure 1. As in the CNLP paper, it is possible that the road from point B to Snowbird and/or the road from point C to Park City will be covered with snow and impassable. However, the condition of those roads can only be observed from points B and C, respectively. For the purposes of the current paper, we also introduce two temporal constraints:

- If the agent skis at Snowbird, it should arrive after 1p.m., because they offer great discounts in the afternoons.

- If the agent skis at Park City instead, it should arrive at point C no later than 11 a.m., because traffic is very heavy afterwards.

In our formalism, we will denote propositions with capital letters from the beginning of the alphabet, e.g. A, B, C. We will denote literals as A, or $\neg A$, and conjunctions of literals as lists of literals, e.g. $AB\neg C$ denotes $A \wedge B \wedge \neg C$.

Definition 3.1. A **label** l is a conjunction of literals, e.g. $l = ABC$.

Definition 3.2. Two labels l_1 , l_2 are **inconsistent** (denoted as $Inc(l_1$, $l_2))$ iff $l_1 \wedge l_2$ is *False*. E.g. $l_1 = AB$ and $l_2 = \neg B\neg C$ are inconsistent. Two labels l_1, l_2 are **consistent** (denoted $Con(l_1, l_2)$) iff $\neg Inc(l1, l2))$. A label l_1 **subsumes** label l_2, (equivalently, l_1 is more specific than l_2), iff $l_1 \Rightarrow l_2$. E.g. if $l_1 = AB\neg C$ and $l_2 = A\neg C$, then l_1 subsumes l_2 (denoted $Sub(l_1, l_2)$)

Definition 3.3. The set of all possible labels defined with respect to a set of propositions P, is the **label universe** of P, P^*.

We are now ready to formally define Conditional Temporal Problems, inspired by the definitions of the TCSP, STP and DTP.

Definition 3.4. A **Conditional Temporal Problem (CTP)** is a tuple $< V, E, L, OV, O, P >$, where:
P is a finite set of propositions A, B, C, ...
V is a set of nodes (interchangeably called variables, time-points, events) {x, y, z, ... }
E is a set of constraints between nodes in V
$L : V \rightarrow P^*$ is a function attaching a label to each node, e.g. $L(n) = AB$
$OV \subseteq V$ is the set of observation nodes
$O : P \rightarrow OV$ is a bijection associating a proposition with an observation node. $O(A)$ is the node that provides the truth-value for proposition A.

CTPs can be interpreted as follows. When a value is assigned to a time point in V, it indicates the time of occurrence of that event. In planning problems, the time of occurrence is the time at which an action is executed.[2] The times assigned to the nodes in a CTP must satisfy the constraints in E. A major difference from other non-conditional temporal problems is that a node $v \in V$ should only be executed if its label's value becomes *True*. At the time of their execution the observation nodes provide the truth-value of propositions, which in turn determine the truth-value of labels. Note that each proposition has an implicit time point associated with it. Suppose that A is a fluent whose value may change over time (for example `battery-level-low`), and further suppose that the value of A needs to be observed at two different times. Then the CTP will include two distinct propositions, e.g. A_{T_1} and A_{T_2}, each associated by the function O to a different observation node. Therefore for each proposition in P there exists a unique observation node.

It is reasonable to assume that in any well-defined CTP there should not be any constraint relating nodes with inconsistent labels, since those nodes will never be executed under the same circumstances. Also, it is reasonable to require that when we execute a node v, we have to know the truth-value of its label $L(v)$. This in turn implies that (i) all nodes observing a proposition in $L(v)$ are executed in all cases in which v is executed, and (ii) they are executed before v. To ensure

[2] If the action has temporal duration, the model includes two nodes for every action: one that denotes the time at which execution of the action begins, and another that denotes the time at which it ends.

these requirements for any node v for which A appears in $L(v)$, we can statically check that $Sub(L(v), L(O(A))$ and add the constraint $O(A) < v$ to the temporal problem definition.

We now define three types of CTPs, which differ in the types of constraints they allow.

Definition 3.5. A *Conditional Simple Temporal Problem* **(CSTP)** is a CTP where the constraints in E are STP-like constraints, i.e. binary constraints (called edges) of the form $(l \leq y - x \leq u)$. We similarly define *Conditional Disjunctive Temporal Problems* **(CDTP)** and *Conditional Temporal Constraint Satisfaction Problems* **(CTCSP)**, by analogy to DTPs and TCSPs.

Example 3.1. Figure 2 shows a CSTP that encodes part of the ski-trip example. (To keep the example small, we omit the part of the plan that involves traveling from point C to Park City.) The vertices V represent three types of events:

- The start and end points of the *go* actions; (go x y) denotes the action of going from location x to location y. A node labeled (go x y)$_S$, i.e. with subscript S, indicates the event of starting the action (go x y); similarly (go x y)$_E$ denotes the event of ending such an action (and arriving at y).

- The observation events; (obs (road x y)) denotes observing the condition of the road from x to y, while located at x. For clarity of presentation, we treat observation steps in our examples as if they were instantaneous, although in general this is not a requirement.

- The special *Start* event, which is associated with a specific arbitrary point in time: in our example, 12a.m. It is used to encode absolute time constraints, for example, that if the agent goes to Snowbird, he should not arrive there until 1p.m. (13 hours after *Start*). In the temporal-reasoning literature, the Start node is usually denoted "TR," for Temporal Reference point.

We will follow the convention that all non-annotated edges in the figures are assumed to have bounds $[0, \infty]$ and the labels of the unlabeled nodes are *True*. There is exactly one observation action (obs (road b s)) that provides the truth-value of A, namely whether the road from b to s is open: thus, $O(A) =$ (obs (road b s)). Notice that this node satisfies the two conditions identified above: It is executed in every context in which steps labeled with A or $\neg A$ are executed (in fact, in this example, it is *always* executed), and it is executed before any of the steps that are labeled with A or $\neg A$ (note the implicit $[0, \infty]$ constraint on the outgoing arcs from it).

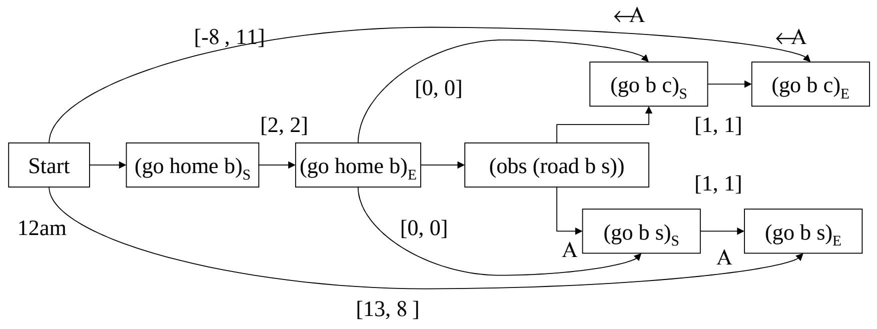


Figure 2. The skiing plan as a Conditional Simple Temporal Problem. All non-annotated edges are assumed [0, ∞] and all non-annotated nodes are assumed labeled *True*.

Definition 3.6. An **execution scenario** s is a label that partitions the nodes in V into two sets V_1 and V_2 : $V_1 = \{n \in V : Sub(s, L(n))\}$ and $V_2 = \{n \in V : Inc(s, L(n))\}$, i.e. the set of nodes that will be executed because s implies their label is *True*, and the set of nodes that will not be executed because s implies their label is *False*. An execution scenario contains all the information (i.e. the value of all necessary propositions) to decide which nodes to execute and which not to execute. We will use **SC** to denote the set of scenarios for a given CTP.

Theorem 3.1. *Any complete truth-assignment to the propositions in P is an execution scenario (we will call it a* **complete scenario***). (See Appendix for proof).*

Definition 3.7. A **scenario projection** of the CTP $< V, E, L, OV, O, P >$ of an execution scenario s, denoted as $\mathbf{Pr(s)}$, is the temporal non-conditional problem $< V_1, E_1 >$, where $V_1 = \{n \in V : Sub(s, L(n))\}$ and $E_1 = \{(v_1, v_2) \in E : v_1, v_2 \in V_1\}$. This will be an STP, DTP, or TCSP depending on the constraints in E. Put simply, the scenario projection of an execution scenario s is the set of nodes of the CTP that will be executed under s and all the constraints among them.

Example 3.2. In our skiing example, there are only two possible execution scenarios: A and $\neg A$, where $\mathbf{Pr(A)}$ includes, intuitively, all the nodes that are executed when the road to Snowbird is clear and the agent skis there, and, similarly, $\mathbf{Pr(\neg A)}$ includes all the nodes that are executed when the road to Snowbird is closed and the agent instead heads towards Park City via point C. For a more interesting example, consider the CSTP in Figure 3, which has nodes $\{TR, y, z, w, u, v,$

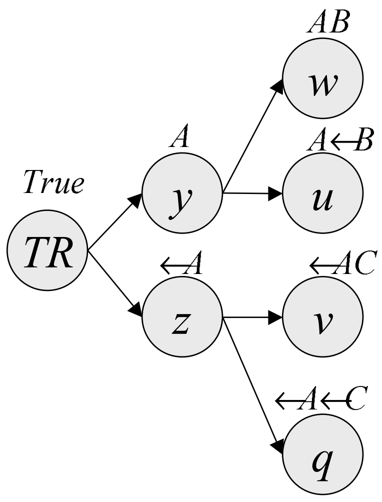


Figure 3. The CTP of the Example 3.2.

$q\}$ with labels $\{True, A, \neg A, AB, A\neg B, \neg AC, \neg A\neg C\}$, respectively. Suppose that $TR = O(A)$ observes A, $y = O(B)$ observes B, and $z = O(C)$ observes C. The CSTP thus has the typical structure of a conditional plan in which TR is initially executed and A is observed; if it is true y is executed, otherwise z is executed; and so on. The execution scenarios AB, ABC and $AB\neg C$ all refer to the same execution, i.e. under each of them, the nodes TR, y, and w, but no other nodes, will be executed.

Definition 3.8. Two execution scenarios are **equivalent execution scenarios** if they induce the same partition on the set of nodes. The "equivalent execution scenarios" relation induces an equivalence class relation **R**. A class in **R** contains all scenarios that are equivalent.

Definition 3.9. A scenario is a **minimum execution scenario** if it contains the minimum number of propositions compared to all other scenarios in its "equivalent execution scenario" class.

In Example 3.2 above, scenarios ABC, $AB\neg C$, and AB all belong to the same equivalence class, with AB being the minimum execution scenario of the class. For this example, there are four minimum scenarios: $\{AB, A\neg B, \neg AC, \neg A\neg C\}$.

4. Notions of Consistency

CTPs have a different notion of consistency than their non-conditional counterparts TCSPs, DTPs, STPs, and most other temporal reasoning problems. In the conditional case, consistency cannot be defined simply as the existence of an assignment to the time-points (nodes) that satisfies the constraints. This interpretation fails to take account of the fact that in the CTP modeling a temporal plan, propositions are usually only observed at execution time. We thus present three notions of consistency, which differ from one another in the assumptions made about when observation information is known.

In the first case, we require a notion of consistency that allows for off-line scheduling, i.e. allows precise times for all events to be determined before execution begins. Here it is reasonable to assume that the scheduling algorithm has no information about the outcome of the observations. Therefore it should schedule the nodes in such a way that the constraints are satisfied no matter how the observations turn out. If such a schedule exists, then we will say that the CTP is Strongly Consistent.

As a second case, consider an agent that plans for a number of initial future states. Each initial state corresponds to a set of initial truth-values of some propositions, i.e. an execution scenario. The specific scenario is unknown at planning time, but it will be known to the execution agent prior to execution. Then, it is necessary for the planning agent to verify that no matter which initial state turns out *True*, the execution agent has a way to execute the plan. If this is possible, the CTP is Weakly Consistent.

The third, most complicated and typical case, assumes that information about the outcome of observations becomes known during execution. Unlike the first case, however, it allows the decisions about the timing of events to be made dynamically at execution time. We will call a CTP Dynamically Consistent if it can be executed so that no matter what the outcome is for the upcoming observations, the current partial solution (i.e. the assignment of values to time-points) can be extended so that all constraints are satisfied.

These notions of consistency are similar to those developed for STPUs, where consistency is defined in terms of *controllability* (Vidal and Fargier, 1999). Essentially, a network is controllable if there is a strategy for executing the timepoints under the agent's control that satisfies all requirements. Three primary levels of controllability had also been identified. In *Strong Controllability*, there is a static control strategy that is guaranteed to work in all situations. In *Weak Controllability*, for all situations there is a "clairvoyant" strategy that works if all uncertain durations are known when the network is executed. And in *Dynamic Controllability*, it is assumed that each uncertain duration becomes known (is observed) just after it has finished, and it requires that an execution strategy depend only on the past outcomes. Strong and Dynamic Controllability have been shown to be tractable (Vidal and Fargier, 1999; Morris et al., 2001), while Weak Controllability is conjectured to be co-NP-complete (Vidal and Fargier, 1999).

Definition 4.1. A **schedule** T of a CTP $< V, E, L, OV, O, P >$ is a mapping $V \rightarrow \Re$, i.e. a time assignment to the nodes in V. We denote with $T(v)$ the time assigned to node v.

Definition 4.2. An **execution strategy** St is a function from the set of scenarios for a CTP to a schedule $St : \mathbf{SC} \rightarrow T$. A **viable execution strategy** is one such that $St(s)$ is a solution to the projection $\mathbf{Pr(s)}$ for each scenario $s \in \mathbf{SC}$.

Definition 4.3. Given a scenario s and a schedule T, for each node x in the projection scenario of s, we can determine the set of all observations performed before time $T(x)$, along with their outcomes. We will call the set of the observation outcomes before time $T(x)$ the **observation history** of x relative to scenario s and schedule T and will denote it as $H(x, s, T)$.

Definition 4.4. A CTP $< V, E, L, OV, O, P >$ is **Weakly Consistent** if there exists a viable execution strategy for it, i.e. every projection $\mathbf{Pr(s)}$ is consistent in the STP/DTP/TCSP sense.

Definition 4.5. A CTP $< V, E, L, OV, O, P >$ is **Strongly Consistent** if there is viable execution strategy St such that, for every pair of scenarios s_1 and s_2, and variable x executed in both scenarios,

$$[St(s_1)](x) = [St(s_2)](x)$$

Thus, an execution strategy that satisfies the definition of Strong Consistency assigns a fixed time to each executable timepoint irrespective of the observation outcomes. The idea behind finding a single schedule is similar to that of finding a conformant plan (Goldman and Boddy, 1996).

Definition 4.6. A CTP $< V, E, L, OV, O, P >$ is **Dynamically Consistent** if there is a viable execution strategy St such that, for every variable x and pair of scenarios s_1 and s_2,

$$Con(s_2, H(x, s_1, St(s_1)) \vee Con(s_1, H(x, s_2, St(s_2)))$$
$$\Rightarrow [St(s_1)](x) = [St(s_2)](x)$$

In the definition above[3], $Con(s_2, H(x, s_1, St(s_1)))$ means that the set of observation outcomes uncovered before x in scenario s_1 forms a label that is still consistent with scenario s_2 at the time at which x is to be performed in s_1. Thus, at time point x, the agent has not yet distinguished between scenarios s_1 and s_2. Therefore it must assign to x the same time in s_1 and s_2. The same arises in the opposite case (x in scenario s_2 while s_1 is still feasible). An execution strategy that satisfies the above definition ensures that the scheduling decisions that are taken (while executing) only use information available from previous observations.

To compare the three notions of consistency, reconsider the CSTP of Figure 2 as defined in Example 3.1. Is the network Strongly Consistent? We can immediately see that, if A is *True* (i.e. the agent observes that the road to Snowbird is open), then it must arrive and then immediately leave point b no sooner than noon, i.e. (go home b)$_E \geq 12$, given the constraints $13 \leq$ (go b s)$_E - Start \leq \infty$, $0 \leq$ (go b s)$_S -$ (go home b)$_E \leq 0$, and $1 \leq$ (go b s)$_E -$ (go b s)$_S \leq 1$. That is, because the agent must not arrive in Snowbird before 1p.m., there is no place to wait at point b, and it takes an hour to get from b to Snowbird, it must arrive at b no earlier than noon. An analogous argument let us deduce that if A is *False* it must arrive at b no later than 10a.m., i.e. (go home b)$_E \leq 10$ Thus, there is no way to construct a schedule for this network without knowing the truth-value of A. Hence, the CTP is not Strongly Consistent.

However, it is Weakly Consistent. To prove this we just have to provide a consistent schedule for each scenario. In this example there are two execution scenarios $s_1 = A$ and $s_2 = \neg A$, which means only the

[3] We point out that our consistency definitions and algorithms would still be valid had we defined labels as any propositional formula on P^*. We chose to restrict labels to conjunctions for illustration purposes. Also, note that although they are symmetric, both disjuncts $Con(s_2, H(x, s_1, St(s_1))$ and $Con(s_1, H(x, s_2, St(s_2)))$ are required for the definition to cover only the set of plans that are actually executable. Here is an example that shows that using only one of the disjuncts is not enough. Consider the CTP with nodes x, y, z, w, $L(z){=}A$, $L(w){=}\neg A$, $L(x){=}L(y){=}True$, and $O(A) = y$, and constraints $x - y \in [-5, 5]$, $z - y = 5$, $w - y = 15$, $z - x = 10$, and $w - x = 10$. If A is *True*, only the schedule $T_1(x) = 0$, $T_1(y) = 5$, and $T_1(z) = 10$ and all of its translations $T_1 + t$ are consistent. If A is *False*, only the schedule $T_2(y) = 0$, $T_2(x) = 5$, and $T_2(w) = 15$ and all of its translations are consistent. T_1 and T_2 define the execution strategy St with $St(A) = T_1$ and $St(\neg A) = T_2$. The CTP is obviously not dynamically consistent since we need to know the value of A before execution in order to decide whether we should follow T_1 or T_2, but A is known only after x has been executed in T_1. However, $H(x, \neg A, T_2) = \neg A$ and so $\neg Con(A, H(x, \neg A, T_2))$. If only this disjunct was used in the definition, the antecedent of the implication would be *False*, and the constraint always satisfied and thus St would satisfy the definition, falsely implying the CTP is dynamically consistent.

truth-value of A discriminates between possible scenarios. One consistent schedule for s_1, which we will call T_1, assigns (go home b)$_S$ a time of 10 a.m. (with all the other time points being directly derivable from that, e.g., (go home b)$_E$ is assigned noon). A consistent schedule for s_2, T_2 assigns (go home b)$_S$ a time of 8 a.m. The execution strategy St, where $St(s_1) = T_1$ and $St(s_2) = T_2$ is viable. So, provided only that the value for A is known before execution starts, the agent simply needs to pick the corresponding schedule T_1 or T_2.

Is the network Dynamically Consistent? In the discussion above we showed that T((go home b)$_E$) must be greater than or equal to 12 if A is observed *True*, and less than or equal to 10 if A is observed to be *False*. In turn, this forces T((go home b)$_S$) to be greater than or equal to 10 if A is *True*, and less than or equal to 8 if A is *False*. If we could observe A before starting out on the journey, i.e. before event (go home b)$_S$, then we could distinguish between the two scenarios, determine which one we fall into, and schedule our departure from home accordingly. However, the problem is set up such that being at point b is a precondition for the observation action. Thus, there is no way to perform the observation, and determine the value of A, "in time" to schedule the departure. The example is not dynamically consistent.

Let us now state an obvious property of the three notions of consistency, that is similar to the corresponding one in STPU:

Theorem 4.1. *Strong Consistency $\Rightarrow$ Dynamic Consistency $\Rightarrow$ Weak Consistency*

5. Strong Consistency

We now present an important property of Strong Consistency for CTPs.

Theorem 5.1. *A CTP $< V, L, E, OV, O, P >$ is Strongly Consistent if and only if the (non-conditional) temporal problem $< V, E >$ is consistent. (See Appendix for proof).*

The implication of the above theorem is that we can perform Strong Consistency checking by using specialized algorithms for non-conditional temporal problems such as IDCP (Chleq, 1995) for STPs, known techniques (Schwalb and Dechter, 1997) for TCSPs, and Epilitis (Tsamardinos, 2001) for DTPs.

5.1. Uses of the Strong Consistency Concept for Planning

A plan that is represented as a strongly consistent CTP can be executed according to a fixed schedule, in the sense that every action it includes has an assigned, specific time. Not all of the actions will occur in every scenario. Thus, the plan is contingent in the same sense as plans generated by CNLP (Peot and Smith, 1992): Certain actions may or may not be executed, depending on the results of execution-time observations. A contingent plan with a fixed schedule should be contrasted with temporally contingent plan, in which the times at which actions are performed also depends upon such observations. Plans with necessary temporal contingencies will be dynamically consistent (as discussed in Section 7) but not strongly consistent.

Thus, Strong Consistency is a restrictive type of consistency, meaning that a CTP might not be Strongly Consistent but nonetheless can be executed. Nevertheless, since we can employ existing algorithms and systems for determining Strong Consistency it is possible to build a temporal and conditional planner using those systems. Such a planner performs a search in the plan space adding appropriate actions, observations and temporal constraints to resolve the conflicts (threats) in the CNLP style. The consistency of the underlying temporal constraints in the CTP representing the current plan can then be determined.

6. Weak Consistency

6.1. Weak Consistency Checking

It is easy to design a brute force algorithm for checking Weak Consistency. The task involves finding a solution to the temporal subproblem $\mathbf{Pr(s_i)}$ for every execution scenario s_i. Thus, the two steps of the algorithm are:

1. Find the set of execution scenarios $\mathbf{SC}$.
2. Check the consistency of the non-conditional problem $\mathbf{Pr(s)}, \forall s \in \mathbf{SC}$.

Let us examine a specific example of the above, on the CSTP of Figure 2. The two projections $\mathbf{Pr(A)}$ and $\mathbf{Pr(\neg A)}$ and all the constraints in these two projections are shown in Figure 4. Consistency can be easily proven in each projection.

We can improve on this algorithm by noticing that $\mathbf{Pr(s_i)} = \mathbf{Pr(s_j)}$ for every two equivalent scenarios s_i and s_j. Thus, we only need to select one scenario s_i from each class in $\mathbf{R}$ of Definition 3.8 and check

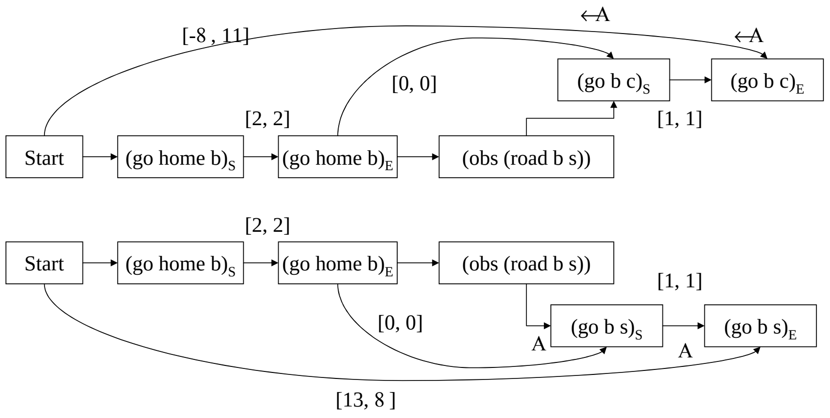


Figure 4. The projected STPs of Figure 2 for all scenarios. $\mathbf{Pr}(\neg\mathbf{A})$ is shown in the top part and $\mathbf{Pr}(\mathbf{A})$ in the bottom part.

the consistency of $\mathbf{Pr}(\mathbf{s_i})$. It might even be desirable to select the minimum execution scenario as the representative of its class. Then the first step of the algorithm is the problem of finding the *set of minimum execution scenarios*. We solve this problem in (Tsamardinos, 2001) where we present an algorithm that calculates this set without explicitly enumerating all possible scenarios. We also prove the complexity result that Weak CSTP Consistency is co-NP-Complete (see Theorem A.1 in the Appendix).

Another way to improve the Weak Consistency checking algorithm is to perform the second step of the algorithm incrementally. For example, when solving the sequence of problems $\mathbf{Pr}(\mathbf{s_1}), \mathbf{Pr}(\mathbf{s_2}), \ldots$ there is often shared computation between problems $\mathbf{Pr}(\mathbf{s_i})$ and $\mathbf{Pr}(\mathbf{s_{i+1}})$. The order of consideration of each $\mathbf{Pr}(\mathbf{s})$ highly influences the amount of computation that can be shared. An algorithm that employs this idea and calculates an appropriate order of incremental consistent checks for the series of $\mathbf{Pr}(\mathbf{s_i})$ is again presented in (Tsamardinos, 2001) (Chapter 6) for the CSTP case.

6.2. Uses of the Weak Consistency Concept for Planning

If a plan is represented as a weakly consistent CTP that is not also dynamically or strongly consistent, this means that there is always a non-contingent plan for any set of observations, but to execute that plan, we must know the observations at the start of execution.

We now suggest a possible use of Weak Consistency for planning purposes, namely in planning architectures that are based on plan-merging. Examples of such architectures are Workflow Management

Systems (Georgakopoulos et al., 1995), PRS (Myers, 1997), the Plan Management Agent (Tsamardinos et al., 2000), and Autominder (Pollack et al., 2002) to name a few. In these systems, there is a library of plan (or workflow) schemata, and whenever a new goal arrives, a plan from the plan library is selected and subsequently merged with the system's existing commitment structure, i.e. the set of (partially) instantiated plans which it has already adopted. The plan to be merged in the context will depend both upon the new goal to satisfy and the current set of commitments. The conditions set by the latter form a "scenario" for which there should exist a corresponding schedule, which makes Weak Consistency relevant. The plan library may include a number of different schemata that achieve a given goal G. A CTP can be used to compactly represent all such schemata. For example, if P_1 achieves goal G when A is *True*, and plan P_2 achieves G when $\neg A$, then we can build a CTP that simultaneously represents both P_1 and P_2 by attaching appropriate labels A and $\neg A$ on the temporal variables in the plans. Now assume that P_1 and P_2 share a large part of their structure and differ only in a few preparatory steps. It is easy to see that the CTP representation can attach the label *True* to all common steps and this way both display the shared structure and remove the redundancy.

As a simple example consider the CTP shown in Figure 5(a), which encodes the four non-conditional plans in Figure 5(b). (Imagine that label A denotes "fuel tank empty" and label B denotes "load over 2000 lbs"). Not only is the CTP encoding more compact, but checking its consistency using efficient Weak Consistent checking algorithms will be faster, by definition, than individually checking the consistency of each the non-conditional temporal plans.

7. Dynamic Consistency

7.1. Dynamic Consistency Checking

We now turn to Dynamic Consistency. In order to distinguish between the execution of the same node in different scenario projections, we use $N(x, s)$ to denote node x in $Pr(s)$. Also, let us denote with $\mathit{Diff}_{s_2}(s_1)$ the set $\{N(v, s_1)\}$ of all nodes v in s_1 that provide observations with outcomes that differ from the corresponding ones in s_2.

To prove Dynamic Consistency of a CTP we need to identify a viable execution strategy St satisfying the conditions of Definition 4.6. The condition $Con(s_2, H(x, s_1, St(s_1)))$ is satisfied if and only if at the time $N(x, s_1)$ is executed, there is no observation node $N(v, s_1)$ in $\mathit{Diff}_{s_2}(s_1)$

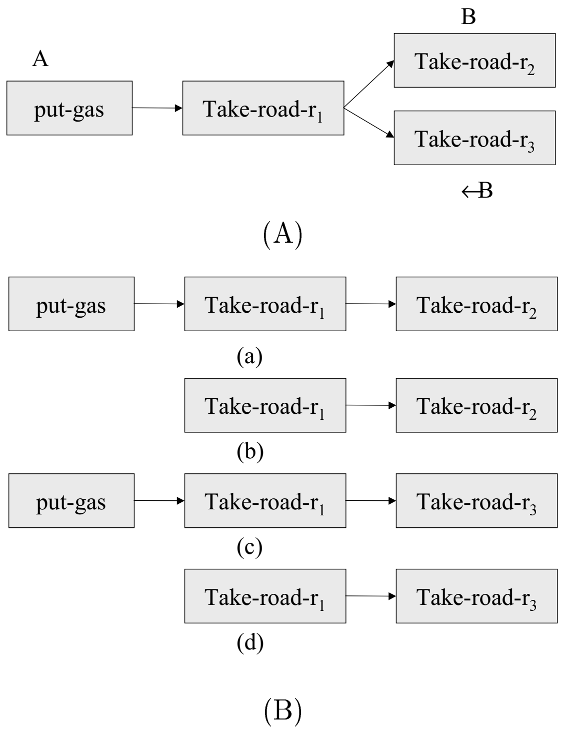


Figure 5. (A) A CTP encoding four different plan schemata. (B) The non-conditional plan schemata represented by the CTP in (A).

that has been executed yet; otherwise, we could distinguish between the two scenarios. This directly translates to the following equation for $Con(s_2, H(x, s_1, St(s_1)))$:

$$Con(s_2, H(x, s_1, St(s_1))) \Leftrightarrow \bigwedge_{N(v,s_1) \in Diff_{s_2}(s_1)} N(x, s_1) \leq N(v, s_1)$$

and similarly for $Con(s_1, H(x, s_2, St(s_2))$. Thus, we can rewritte the condition in Definition 4.6 as:

$$\{ \bigwedge_{N(v,s_1) \in Diff_{s_2}(s_1)} N(x, s_1) \leq N(v, s_1)\} \bigvee \{ \bigwedge_{N(v,s_2) \in Diff_{s_1}(s_2)} N(x, s_2) \leq N(v, s_2)\} \Rightarrow N(x, s_1) = N(x, s_2) \quad (1)$$

On the left-hand side of the implication, we have simply replaced each of the two Con conditions from Definition 4.6 with a conjunction over

```
DynConsistency(CTP $Ctp$)
1. DTP $D :< V, C > = < \cup_i V_i, \cup_i E_i >$,
              where $Pr(s_i) = < V_i, E_i >$ are the projected scenarios.
2. For each pair of scenarios $s_1, s_2$
3.        For each node $v$ that appears in both $s_1, s_2$
4.              $C = C \wedge DC(v, s_1, s_2)$
5.        EndFor
6. EndFor
7. If $D$ is consistent, return Dynamic-Consistent
8. Else return non-Dynamic-Consistent
```

Figure 6. The Dynamic Consistency algorithm

times of observation nodes; the right-hand side of the implication has remained unchanged.

The main idea behind the consistency checking algorithm is to view the above condition of the definition as a (disjunctive) constraint between nodes $N(x, s)$: These together with the set of all nodes $N(x, s)$ for every node x and scenario s of the original CTP define a new temporal problem, namely a DTP D. With this reformulation, an execution strategy St defines a schedule T of D and vice versa by setting $[St(s)](x) = T(N(x, s))$. The aim is to add appropriate constraints to D so that a solution schedule of D will correspond to a Dynamic execution strategy and vice versa. So, in addition to the constraints resulting from Equation (1), we need to impose on the nodes of D all the constraints in every projection $\mathbf{Pr(s)}$. Then, every solution to D will satisfy both the constraints in each projection (thereby guaranteeing that the corresponding strategy will be viable), and the Dynamic Consistency conditions. These ideas lead to the design of the algorithm in Figure 6, where $\mathbf{DC(x, s_1, s_2)}$ (called a DC constraint) is used as a shorthand of Equation (1) above.

Let us trace the algorithm on a specific example such as the CSTP of Figure 2 where $O(A)$ = (obs (road b s)). Line 1 of the algorithm creates a DTP with all the nodes and edges in the two projections $\mathbf{Pr(A)}$ and $\mathbf{Pr(\neg A)}$. The result is shown in Figure 7 (with some additional constraints explained below). To simplify the equations we renamed the nodes (go home b)$_S$, (go home b)$_E$, (obs (road b s), (go b s)$_S$, and (go b s)$_E$ as x, y, z, w, and v. We notice that $Diff_{\neg A}(A) = \{N(z, A)\}$ and $Diff_A(\neg A) = \{N(z, \neg A)\}$ and so the DC for *Start*, $DC(Start, A, \neg A)$, is

$$N(Start, A) \leq N(z, A) \vee N(Start, \neg A) \leq N(z, A)$$

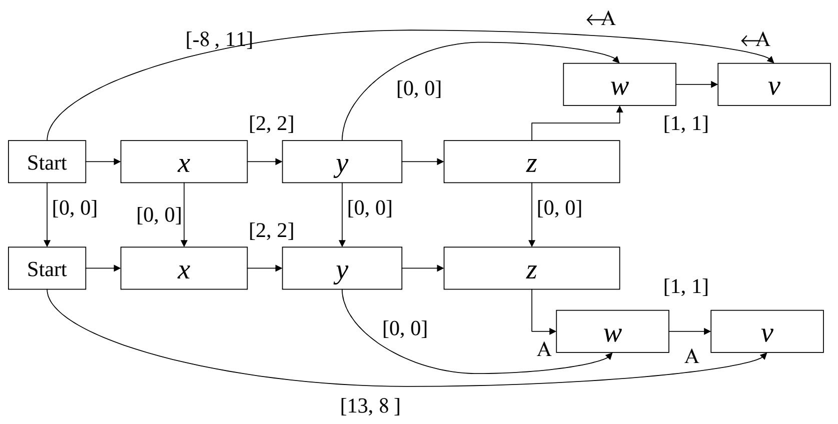


Figure 7. The DTP created by the Dynamic Consistency algorithm, including $\mathbf{Pr}(\neg \mathbf{A})$ (top part), $\mathbf{Pr}(\mathbf{A})$ (bottom part), and DC constraints between them.

$$\Rightarrow N(Start, A) = N(Start, \neg A).$$

Since *Start* is before z in both cases, $N(Start, A) = N(Start, \neg A)$. Similarly, we find that $N(x, A) = N(x, \neg A)$, $N(y, A) = N(y, \neg A)$, and $N(z, A) = N(z, \neg A)$. For nodes w and v the antecedent of the DC implication is always *False* (they occur after z in both scenarios) and so the DC constraint is already satisfied. The result after adding all the DC constraints in Line 4 of the algorithm is shown in Figure 7. The resulting DTP is actually an STP and it is inconsistent, indicating that the original CTP is not Dynamically Consistent.

Suppose instead that the constraints ordering z after y are dropped and z can now be executed any time after *Start*. In particular, other constraints permitting, it could be scheduled before x and y. Then, the DC constraint for x specifies that either x occurs before z in which case $N(x, A) = N(x, \neg A)$, **or** it occurs after z in both scenarios. Thus, the DC constraints are disjunctive in general (recall that $a \Rightarrow b$ is equivalent to $\neg a \vee b$). In case where z is allowed to be executed before x and y the CTP is Dynamically Consistent. Semantically this corresponds to the case where we observe whether the road from b to s is open before we leave home. In that case, we can decide when to start the trip for each different scenario.

Notice that in Equation (1) if the observation nodes in all scenarios are constrained to be ordered with respect to each other, then the conjunctions over all $N(v, s_i) \in \mathit{Diff}_{s_j}(s_i)$ can be substituted with the single minimum of the order. Then Equation (1) becomes

$$N(x, s_1) \leq N(n, s_1) \vee N(x, s_2) \leq N(m, s_2) \Rightarrow N(x, s_1) = N(x, s_2) \quad (2)$$

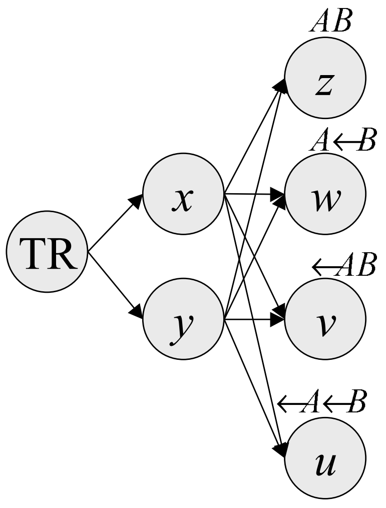


Figure 8. A CTP with two observation nodes unordered with respect to each other.

n and m being the nodes for which $N(n, s_1), N(m, s_2)$ are minimum in the order of the nodes in $\mathit{Diff}_{s_2}(s_1)$ and $\mathit{Diff}_{s_1}(s_2)$ respectively. In general, observation nodes that are constrained to be scheduled after others in the sets $\mathit{Diff}_{s_2}(s_1)$ and $\mathit{Diff}_{s_1}(s_2)$ are ruled out of the conjunctions in Equation (1).

A more complicated example is shown in Figure 8 where two observation nodes $O(A) = x$ and $O(B) = y$ are unordered with each other so Equation (1) cannot be simplified to the form of Equation (2). Let us consider node x and scenarios $s_1 = AB$ and $s_2 = A\neg B$. Then, $\mathit{Diff}_{s_2}(s_1) = \{N(y, s_1)\}$ while $\mathit{Diff}_{s_1}(s_2) = \{N(y, s_2)\}$. Thus, $DC(x, s_1, s_2)$ is the constraint $N(x, s_1) \leq N(y, s_2) \vee N(x, s_2) \leq N(y, s_2)$ $\Rightarrow N(x, s_1) = N(x, s_2)$. If we decide to perform the observation for A first, i.e. $x < y$, then $DC(x, s_1, s_2)$ becomes $N(x, s_1) = N(x, s_2)$. The resulting STP is shown in Figure 9(a). In the other case (where we defer the observation of A) we end up with the STP in Figure 9(b) where there is no constraint between $N(x, s_1)$ and $N(x, s_2)$. Since the observations are unordered, the DC constraints are disjunctive and represent in a DTP both of these alternative STPs of (a) and (b). The original CTP is Dynamically Consistent, if and only if one of these alternatives is consistent.

It is important to note that we reduced the consistency checking problem to a DTP because DTPs can represent n-ary disjunctive constraints. TCSPs and STPs do not allow this, and thus would not satisfy our requirements. Constraints of this type are typical of Dynamic Consistency in CTPs and make the problem intractable in general.

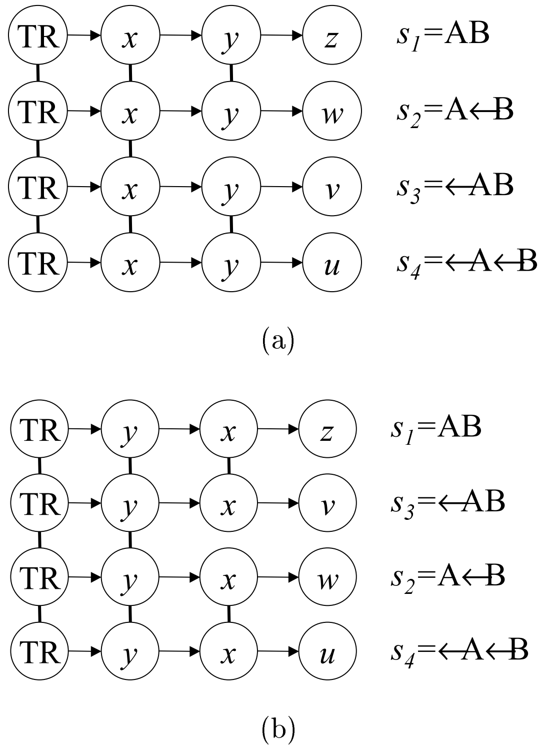


Figure 9. The STP projections with DC constraints for each order of observations. Directed edges are assumed $[0, \infty]$ and undirected edges denote $[0, 0]$ constraints.

This contrasts with Dynamic Controllability in STPUs in which constraints can be reduced to simple STP constraints, hence allowing the design of polynomial-time solution algorithms. Thus, DTP solving algorithms such as Epilitis (Tsamardinos, 2001), which include a number of highly effective heuristic pruning techniques, will have a direct effect on Dynamic Consistency checking in CTPs.

Regarding the complexity of the DTP D, notice that D contains $O(|V|\,|SC|)$ variables, where V is the set of variables in the CTP and SC the set of (minimal) scenarios whose number is in the worst case exponential to the number of propositions $|P|$. In the worst case the constraints are disjunctive and, when put in Conjunctive Normal Form, may create an exponential number of disjunctive clauses. Nevertheless, some structural properties of the CTP help in reducing the complexity.

First, as we have noted above, when the observation nodes are ordered with respect to each other in every scenario, the DC constraints are given by Equation (2) which is a great simplification over Equation (1). Additionally, if each node is ordered with respect to every observation node in all scenarios, then the antecedent of each DC constraint can be statically checked. In this case, the DC constraint either becomes $N(x, s_1) = N(x, s_2)$ or it is already satisfied. For instance a CSTP can then be made equivalent to a larger STP, since no disjunctive constraints are added to the problem, which allows very efficient Dynamic Consistency checking.

7.2. Uses of Dynamic Consistency Concept for Planning

Dynamic Consistency checking can be used to build a temporal and conditional planner. It is easy to see that by appropriately modifying the CNLP algorithm (Peot and Smith, 1992) it is possible to allow the simultaneous representation of and reasoning with quantitative temporal constraints and conditional branches. When a temporal constraint $x < y$ is added to the CTP representing the conditional plan (e.g. to resolve a conflict), the Dynamic Consistency algorithm can determine whether the resulting plan is executable. Moreover, this notion of "executable" goes beyond that of traditional planning system, because it allows for observations to be made at execution time in plans in which timing constraints depend on observation outcomes. Dynamic Consistency checking can also support the merging of such richly expressive plans at execution time, e.g. to handle new goals that arise during execution (Tsamardinos, 2001)(Chapter 7).

In order to execute a Dynamically Consistent plan we can instead execute the DTP D to which we reduced the problem. Notice that we should execute only one node $N(x, s_i)$ for every scenario s_i since they semantically correspond to the same event and the same CTP node. Of course, the algorithm guarantees that $N(x, s_i) = N(x, s_j)$ in all appropriate cases and avoids confusion. We can identify at least three ways D can be executed: (i) We compute a solution to D and execute that. This is the least flexible approach since it commits to a specific schedule (solution) of D. (ii) We find and flexibly execute a consistent component STP of D. Consistent components STPs are returned by DTP solvers such as Epilitis (Tsamardinos, 2001) and can be flexibly executed with algorithms such as in (Tsamardinos et al., 1998). (iii) We flexibly execute the DTP directly, retaining all possible scheduling flexibility, using the algorithm in (Tsamardinos et al., 2001).

Finally, we note that because typical conditional plans satisfy both of the conditions mentioned at the end of the previous subsection, the performance of the DC algorithm during plan construction and merging is likely to be higher than in the general case. In addition, conditional planners generate plans where the number of distinct execution scenarios is linear in the number of propositions. We suspect that in this case the Dynamic Consistency algorithm we presented is actually polynomial in the number of original CTP variables and propositions (Tsamardinos, 2001)(Chapter 6). We intend to formalize these ideas on performance improvements in our future work.

8. Improved Conditional Planning

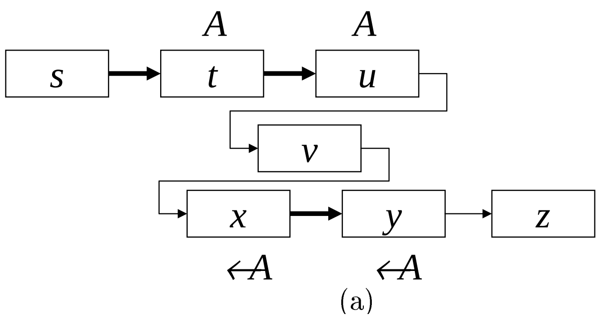
In the previous section, we noted that by using a Dynamic Consistency algorithm, one can extend traditional conditional planners to support quantitative temporal constraints. It is essential to manage those constraints; if they are ignored, then the planner risks generating incorrect plans. For instance, a CNLP-style planner would generate a conditional plan for our skiing example if it simply ignored the two additional temporal constraints (either arrive at Snowbird after 1 p.m., or else arrive at point C on the way to Park City before 11 a.m.). But such a plan would be useless, because, as we have already seen, given the temporal constraints the plan is dynamically inconsistent and there is no way of executing it.

Of course, the traditional conditional planners (Peot and Smith, 1992; Pryor and Collins, 1996; Onder and Pollack, 1999) were not designed to deal with quantitative temporal constraints. But they do perform a limited form of temporal reasoning, in order to deal with ordering constraints, and it turns out that even for plans with only ordering constraints, there are clear advantages to using the dynamic consistency approach.

CNLP propagates context information only along causal links and conditioning links, but not along ordering constraints. We assume that this choice was made so that if a step x with context *True* is promoted after a step y of context A, then the context of x remains *True* and so x can be reused to provide causal links to steps in other contexts, thereby potentially reducing the amount of planning required and resulting in smaller plans.

Nevertheless, this method might reject valid (i.e. executable) plans. An example is shown in Figure 10(a). The bold edges correspond to causal links, and the lighter edges denote ordering constraints added by threat resolution. We suppose that step v clobbers both the causal link $t \rightarrow u$ and the causal link $x \rightarrow y$; hence v has been promoted and demoted respectively to resolve the conficts. We further suppose that z clobbers the latter causal link and has been promoted after y.

When CNLP checks whether an ordering constraint exists between a pair of nodes s and z, it essentially computes the transitive closure and determines whether $s < z$ holds. Since the context information is



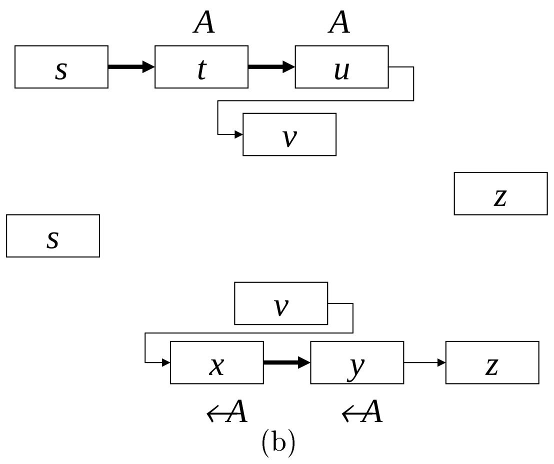


Figure 10. (a) A CNLP plan whose handling by CNLP reasoning falsely induces that s is necessarily before z. (b) The two projections of the plan: z is allowed before s in both.

ignored in this calculation, CNLP essentially calculates Strong Consistency of the induced CTP. However, in Figure 10(b) the two projections of the plan are shown, and it is easy to see that s and z are actually unordered with respect to each other.

As already mentioned in Section 5, Strong Consistency is a restrictive type of consistency and plans that are executable (i.e. Dynamically Consistent) might not be Strongly Consistent. Thus, *CNLP is not complete and it might reject valid plans*, unlike what is conjectured in the original CNLP paper. In the above example, if z is ordered before s and this is the only valid plan, CNLP will reject it as inconsistent even though it is Dynamically Consistent (assuming A is observed before this portion of the plan).

9. Related Approaches and Conclusions

As far as we know only two approaches might be compared to our work. The first paper by Schwalb et al. (Schwalb et al., 1994) separates propositional and temporal reasoning, addressing expressive propositional and temporal constraints, to process deduction and hypothetical reasoning on knowledge bases. The authors define a "Conditional Temporal Network" model, in which some constraints are dependent on a condition and are only used if that condition is *True*. The aim is to make queries in the base such as "is formula F consistent with the current constraints?". Although such a model may be seen as a general logical framework for doing conditional temporal reasoning, it is insufficient for our purpose for two reasons. First, the approach is static and does not deal with the dynamic aspects of plan execution: the time at which a condition is known to be *True* or *False* is not consider, which for planning purposes is crucial. Second, unlike Weak Consistency, which determines whether all scenarios are consistent, they determine whether there is at least one consistent scenario. This is sufficient when processing queries on a knowledge base, when one interpretation is searched for, but in our planning context that would only mean there exists one unique scenario in which the plan will not fail.

The second and more interesting paper is that of Barber (Barber, 2000), which combines quantitative temporal constraints and alternative contexts in a kind of networks that we will call BarN[4]. Barber defines a temporal problem where constraints (instead of nodes) are annotated with a label (in his terminology a context). Consistency in a BarN corresponds to Weak Consistency in a CTP.

A primary difference between BarNs and CTPs is thus that the former is based on conditional constraints while the latter is based on conditional events. In the Appendix (Theorem A.2) we show that we can use the latter to represent the former and thus our formalism is at least as general as Barber's. Because contexts in BarNs are associated with labels, not nodes, the translation from conditional planning is not as clear as with CTPs, which very naturally associate an observation with a node and attach appropriate labels to subsequent nodes. CTPs can then readily check various forms of consistency, using the techniques described earlier. In contrast, with a BarN representation, the planner has to construct the context hierarchy itself given the observations. Perhaps the most important ramification of this implicit treatment of observations is that the notion of Dynamic Consistency cannot be

[4] BarN denotes Barber Networks.

defined for BarNs. This is because the truth value of the contexts is not associated with a particular time-point.

Unlike BarNs, our new Conditional Temporal Problem formalism is geared towards planning and execution purposes. It is a constraint-based formalism for temporal reasoning in the face of uncertain–or contingent–events, and we have described its usefulness for conditional planning. There are many avenues for future research, of which we highlight a few. CTPs deal with temporal uncertainty arising from the outcome of observations, while STPUs handle uncertainty regarding the timing of uncontrollable events. Obviously, a hybrid model and algorithms that handle both sources of uncertainty would be highly desirable. We are also working on identifying minimal structural requirements for CTPs that will enable polynomial-time Dynamic Consistency algorithms. In parallel, we are also investigating efficient Weak Consistency algorithms.

Acknowledgements

Work on this project by the first and third authors has been partially supported by the United States Air Force Office of Scientific Research (F49620-01-1-0066), by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Office, Air Force Materiel Command, USAF (F30602-00-2-0621), by the National Science Foundation (IIS-0085796), and by Andrew Mellon Predoctoral Fellowship. The views and conclusions herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFOSR, DARPA, AFRL, or the U.S. Government. We would also like the anonymous reviewers for their constructive comments.

References

Armando, A., C. Castellini, and E. Giunchiglia: 1999, 'Sat-Based Procedures for Temporal Reasoning'. In: *5th European Conference on Planning (ECP-99)*.

Barber, F.: 2000, 'Reasoning on Interval and Point-based Disjunctive Metric Constraints in Temporal Contexts'. *Journal of Artificial Intelligence Research* **12**, 35–86.

Bessiere, C.: 1999, 'Non-Binary Constraints'. In: *Principles and Practice of Constraint Programming (CP'99)*. Springer. Alexandria, Virginia, USA.

Chleq, N.: 1995, 'Efficient Algorithms for Networks of Quantitative Temporal Constraints'. In: *Proceedings of the Workshop CONSTRAINTS'95*. pp. 40–45.

Dechter, R., I. Meiri, and J. Pearl: 1991, 'Temporal Constraint Networks'. *Artificial Intelligence* **49**, 61–95.

Georgakopoulos, D., M. Hornick, and A. Sheth: 1995, 'An Overview of Workflow Management: From Process Modeling to Workflow Autonomation Infastructure'. *Distributed and Parallel Databases* **3**, 119–153.

Goldman, R. P. and M. S. Boddy: 1996, 'Expressive planning and explicit knowledge'. In: *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems*. pp. 110–117.

Laborie, P. and M. Ghallab: 1995, 'Planning with Sharable Constraints'. In: *Proceedings of the 14th International Joint Conference on A.I. (IJCAI-95)*. Montreal (Canada).

Mackworth, A. and E. Freuder: 1985, 'The complexity of some polynomial network consistency algorithms for constraint satisfaction problems'. *Artificial Intelligence* **25(1)**, 65–74.

Morris, P., N. Muscettola, and T. Vidal: 2001, 'Dynamic Control Of Plans With Temporal Uncertainty'. In: *Proceedings of the 17th International Joint Conference on A.I. (IJCAI-01)*. Seattle (WA, USA), Morgan Kaufmann, San Francisco, CA.

Muscettola, N., P. P. Nayak, B. Pell, and B. C. Williams: 1998, 'Remote Agent: To Boldly Go where No AI System has Gone Before'. *Artificial Intellience* **103**, 5–47.

Myers, K. L.: 1997, 'Procedural Reasoning System: User's Guide'. Technical report, SRI International.

Oddi, A. and A. Cesta: 2000, 'Incremental Forward Checking for the Disjunctive Temporal Problem'. In: *European Conference on Artificial Intelligence (ECAI-2002)*.

Onder, N. and M. E. Pollack: 1999, 'Conditional, Probabilistic Planning: A Unifying Algorithm and Effective Search Control Mechanisms'. In: *Proceedings of the 16th National Conference on Artificial Intelligence*. pp. 577–584.

Peot, M. and D. E. Smith: 1992, 'Conditional Nonlinear Planning'. In: *Proceedings of the First International Conference on AI Planning Systems (AIPS-92)*. College Park, MD, pp. 189–197.

Pollack, M. E., C. McCarthy, S. Ramakrishnan, I. Tsamardinos, L. Brown, S. Carrion, D. Colbry, C. Orosz, and B. Peintner: 2002, 'Autominder: A Planning, Monitoring, and Reminding Assistive Agent'. In: *7th International Conference on Intelligent Autonomous Systems*.

Pryor, L. and G. Collins: 1996, 'Planning for Contingencies: A Decision-Based Approach'. *Journal of Artificial Intelligence Research* **4**, 287–339.

Schwalb, E. and R. Dechter: 1997, 'Processing Disjunctions in Temporal Constraint Networks'. *Artificial Intelligence* **93**, 29–61.

Schwalb, E., K. Kask, and R. Dechter: 1994, 'Temporal Reasoning with Constraints on Fluents and Events'. In: *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, Vol. 2. Seattle, Washington, USA, pp. 1067–1072, AAAI Press/MIT Press.

Smith, D., J. Frank, and A. Jónsson: 2000, 'Bridging the gap between planning and scheduling'. *Knowledge Engineering Review* **15(1)**.

Stergiou, K. and M. Koubarakis: 2000, 'Backtracking algorithms for disjunctions of temporal constraints'. *Artificial Intelligence* **120**, 81–117.

Tsamardinos, I.: 2001, 'Constrained-Based Temporal Reasoning Algorithms with Applications to Planning'. Ph.D. thesis, University of Pittsburgh.

Tsamardinos, I., P. Morris, and N. Muscettola: 1998, 'Fast Transformation of Temporal Plans for Efficient Execution'. In: *Proceedings of the 15th National Conference on Artificial Intelligence*.

Tsamardinos, I., M. E. Pollack, and P. Ganchev: 2001, 'Flexible Dispatch of Disjunctive Plans'. In: *6th European Conference in Planning.*

Tsamardinos, I., M. E. Pollack, and J. F. Horty: 2000, 'Merging Plans with Quantitative Temporal Constraints, Temporally Extended Actions, and Conditional Branches'. In: *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling.*

Vidal, T. and H. Fargier: 1999, 'Handling contingency in temporal constraint networks: from consistency to controllabilities'. *Journal of Experimental & Theoretical Artificial Intelligence* **11**, 23–45.

Appendix

Theorem 3.1. *Any complete assignment to the propositions in P is an execution scenario.*

Proof. Let s be a complete truth-assignment to the propositions in P, l be a label and $p_1 \dots p_n$ be the propositions that appear in l (either positive or negated). The set of p_i will also appear in s since s is complete. If any p_i appears with different sign in s and l then s and l are inconsistent. Otherwise, if all p_i appear with the same sign in s and l then s subsumes l. So, every node, no matter what its label is, will either belong to V_1 or V_2 in Definition 3.6. $\square$

Theorem 5.1. *A CTP $< V, L, E, OV, O, P >$ is Strongly Consistent if and only if the (non-conditional) temporal problem $< V, E >$ is consistent.*

Proof. The theorem states that we can determine Strong Consistency by ignoring the label and the observation information of the nodes in the CTP and just calculate consistency as we would for an STP, TCSP, or DTP depending on the kind of constraints in E.

"$\Leftarrow$" Suppose that the CTP is Strongly Consistent. Then we will show that the temporal problem $< V, E >$ is also consistent. Let T be a schedule of all nodes, such that $T(x) = [St(s_i)](x)$, where s_i some scenario where x is executed. $T(x)$ is a function because (i) x appears in at least one scenario (or it can be removed from the CTP), and (ii) a Strong execution strategy specifies a unique value to every $[St(s_i)](x)$ for all scenarios s_i where x appears. Because St is viable, T satisfies the constraints in all $\mathbf{Pr(s_i)} =< V_i, E_i >$, for every scenario s_i. Since T satisfies the constraints in every set E_i it satisfies the constraints in their union $\cup_i E_i = E$. Thus, T is a solution to $< V, E >$ and so the latter is consistent.

"$\Rightarrow$" Suppose that the temporal problem $< V, E >$ is consistent; we will prove that the CTP $< V, E, L, OV, O, P >$ is Strongly Consistent.

Let T be any solution of $< V, E >$ (it has to have at least one since it is consistent). For every s_i, T is also a solution of $\mathbf{Pr(s_i)} =< V_i, E_i >$ (ignoring any irrelevant assignments $T(x)$ where x does not appear in V_i) since $E_i \subseteq E$. The execution strategy $St(s_i) = T$ is viable (since T is a solution to every $\mathbf{Pr(s_i)} =< V_i, E_i >$) and also $[St(s_i)](x) = [St(s_j)](x) = T(x), \forall x$ as the definition of Strong Consistency requires. □

Theorem A.1. *Weak CSTP Consistency checking is co-NP-complete.*

Proof. We will prove the result by translating in polynomial time and space a SAT problem to the co-problem of checking Weak Consistency, the co-problem being finding a scenario s_i such that $\mathbf{Pr(s_i)}$ is inconsistent. Specifically, we will create a CSTP given a SAT problem such that the SAT problem has a solution, if and only if there is a scenario s_i such that $\mathbf{Pr(s_i)}$ is inconsistent.

Given the SAT problem with Boolean variables $B = \{x, \ldots, y\}$ and clauses of the form $C_i = (x \vee \ldots \vee y \vee \neg z \ldots \neg w)$, $i = 1 \ldots K$ we create a CSTP $< V, E, L, ON, O, P >$ as follows: The set of propositions is $P = B = \{x, \ldots, y\}$. For each clause $C_i = (x \vee \ldots \vee y \vee \neg z \vee \ldots \vee \neg w)$ and each variable appearance x or $\neg x$ in C_i we create a time-point X that we include in V, with label $L(X) = x$ or $L(X) = \neg x$ respectively. Let us denote with $Clause(C_i)$ the nodes of the CSTP that were included because of C_i . Since we are checking for Weak Consistency it does not matter which nodes are observation nodes. The last thing to define are the constraints between the nodes. There is an constraint between a variable $X \in Clause(C_i)$ to each variable Y with consistent label in $Clause(C_{(i+1) mod K})$: $Y - X = -1$ (we will drop the $modK$ clause in the rest of the proof for clarity; just remember that the nodes in the last $Clause(C_i)$ are connected to the nodes in the first $Clause(C_1)$). The translation is obviously linear in the number of SAT variables and linear in the number of clauses of the SAT problem.

Figure 11 illustrates the proof concept. It presents an example, by showing the resulting CSTP from the SAT problem $(x \vee y \vee z) \wedge (x \vee \neg y \vee z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (\neg y \vee z)$. The labels of each node appear on its top right corner. Notice that there are three propositions in the CSTP for the three SAT variables x, y, z that appear in the labels, either as positive or negative literals, and eleven CSTP nodes one for each appearance of a variable in any clause. The nodes in the CSTP are arranged in columns corresponding to $Clause(C_i), i = 1 \ldots 4$ and are named with the variable of the corresponding proposition and the index i . The edges are connected from a node in $Clause(C_i)$ to all the nodes in $Clause(C_{i+1})$. The order of appearance of clauses in the SAT problem is arbitrary. Also recall that all the edges from a node X to

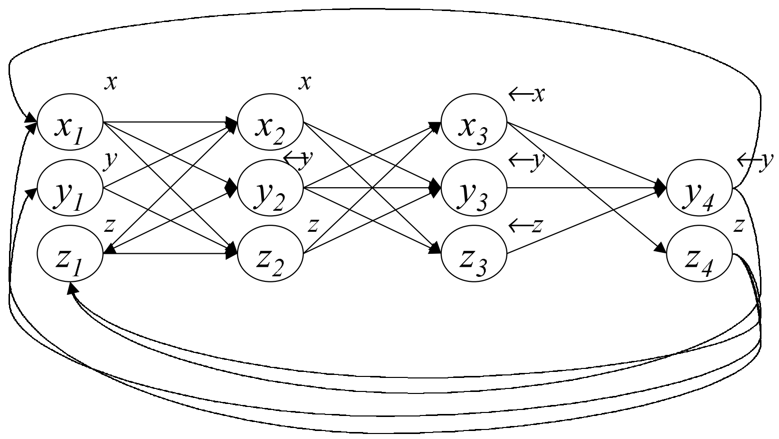


Figure 11. The CSTP resulting from translating a simple example TSAT problem.

a node Y correspond to the constraint $Y - X = -1$ not shown in the figure for clarity. Notice also that there are no edges between nodes with inconsistent labels, e.g. x and $\neg x$.

Let us assume that the SAT problem has a solution $\{x = True, \ldots, y = True, z = False, \ldots, w = False\}$. Since this solution makes *True* at least one variable in a clause, it will make *True* the label of at least one CSTP time-point within $Clause(C_i)$. We will call a time-point whose label becomes *True* in $Clause(C_i)$ L_i (there may be more than one). Notice that each L_i has to have an edge to L_{i+1} because it cannot be the case that L_i has a label x and L_{i+1} a label $\neg x$ by the way the SAT solution is constructed (it never assigned x both *True* and *False* at the same time). Thus the set $\{L_i, i = 1 \ldots K\}$ forms a negative cycle (with weight $(-1) \times K$). There must be at least one scenario s that makes all the nodes in $\{L_i, i = 1 \ldots K\}$ *True*. Namely, let s be the complete scenario that corresponds to the SAT solution (s is a scenario by Theorem 3.1). In other words, $\mathbf{Pr(s)}$ is an inconsistent projected STP. In the above example, the SAT solution $\{x = True, y = False, z = True\}$ makes the labels of the CSTP nodes $x_1, x_2, y_2, y_3, y_4, z_1, z_2$,and z_4 *True* and all the rest *False*. The former set forms at least one negative cycle, e.g. $\{x_1, x_2, y_3, z_4\}$. This completes the proof that if the SAT problem has a solution, the CSTP is not Weakly Consistent.

We will now prove the converse, namely that if the SAT problem has no solution, then the CSTP is Weakly Consistent. Take any complete assignment to the SAT variables. Any such assignment also corresponds to a scenario of the CSTP (by Theorem 3.1). If SAT has no solution, for every such assignment/complete scenario s there is at least one clause C_i that is not *True*, or in other words, all the SAT literals of C_i have to be *False* too. Thus, all the time-points of $Clause(C_i)$

are inconsistent (not executed) under scenario s. But, by the way the CSTP is constructed, every cycle (negative or not) has to go through all clauses. Since no time-point in *Clause*(C_i) becomes *True* under s , there can be no negative cycle in $\mathbf{Pr}(\mathbf{s})$ for any scenario s.

The above argument shows that checking Weak CSTP Consistency is co-NP-hard. Since checking if an STP is consistent is a polynomial problem, co-Weak Consistency is also in co-NP and thus the problem is co-NP-complete.

□

Theorem A.2. *Every (conditional) constraint of the form* $l_1 \leq x_1 - y_1 \leq u_1 \vee \ldots \vee l_k \leq x_k - y_k \leq u_k$ *that should hold only when label* l *is True, can be represented with only conditional events.*

Proof. For any constraint that we want to represent of the form $l_1 \leq x - y \leq u_1 \vee l_2 \leq s - t \leq u_2$ with condition (label) l, we create the dummy nodes w, z, u, v all having label l. We then insert constraints requiring that the pairs of time-points $\{x, w\}$, $\{y, z\}$, $\{s, u\}$, and $\{t, v\}$ co-occur (e.g. $0 \leq x - w \leq 0$), and we also add the (unconditional) constraint $l_1 \leq w - z \leq u_1 \vee l_2 \leq u - v \leq u_2$. This way when l is *True*, nodes w, z, u and v will be executed and, because they co-occur, the original (conditional) constraint on nodes x, y, s and t will be imposed. □

APPENDIX C

Planning Technology for Intelligent Cognitive Orthotics

Martha E. Pollack
Computer Science and Engineering
University of Michigan
Ann Arbor, MI 48103
pollackm@eecs.umich.edu

Abstract

The aging of the world's population poses a challenge and an opportunity for the design of intelligent technology. This paper focuses on one type of assistive technology, cognitive orthotics, which can help people adapt to cognitive declines and continue satisfactory performance of routine activities, thereby potentially enabling them to remain in their own homes longer. Existing cognitive orthotics mainly provide alarms for prescribed activities at fixed times that are specified in advance. In contrast, we describe Autominder, a system we have designed that uses AI planning and plan management technology to carefully model an individual's daily plans, attend to and reason about the execution of those plans, and make flexible and adaptive decisions about when it is most appropriate to issue reminders. The paper concentrates on one of Autominder's three main components, the Plan Manager; other papers in this volume describe its other components (Colbry, Peintner, & Pollack 2002; McCarthy & Pollack 2002).

Introduction

The world's population is aging. The trend in the United States is typical of many industrialized countries. Figures 1 - 3 present populations pyramids based on U.S. census data from 2000, and projections for 2025 and 2050, respectively (Census 2000). Within a population pyramid, each horizontal bar represents the percentage of U.S. residents in a five-year age cohort: the bottom bar represents people aged 0 to 5; the bar above that represents people aged 5 to 10; and so on, up to the topmost bar, which represents people over the age of 100. The population in each age cohort is further divided into males, to the left of the midline, and females, to the right. Historically, the shape of such graphs is pyramidal, as there are more young people than older people.

As can be seen, in 2000, there is a significant bulge in the 25-40 year old cohorts, representing the post-war baby boom, but the basic shape remains pyramidal, with many more people under the age of 60 than people over 60. But by 2025, the pyramid has flattened out, with an increasing proportion of people over 60, and the trend that continues in the 2050 projection.

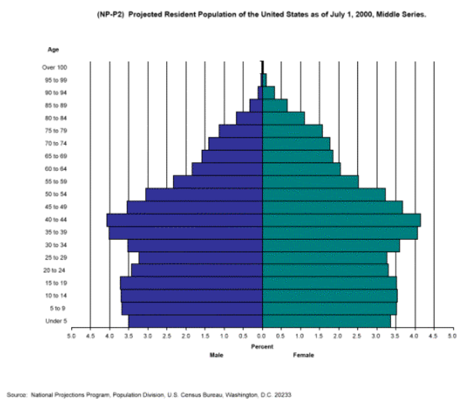


Figure 1: Population Pyramid for the United States in 2000

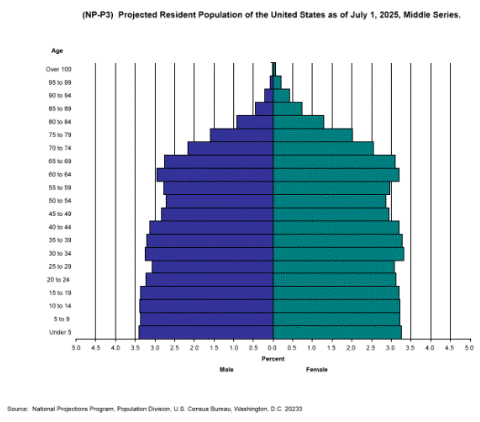


Figure 2: Population Pyramid for the United States in 2025

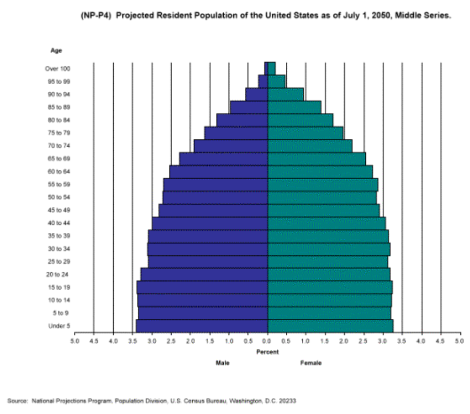


Figure 3: Population Pyramid for the United States in 2050

According to the United Nations Population Division, every region of the world is undergoing a similar demographic transition. In 2000, 606 million people, or approximately 10% of the world's population, were over 60; by 2050, this percentage is expected to double, to 2 billion people, or 21.4% of the population. Even more dramatic will be the increase the percentage of people over 80, often called the "oldest old". Today there are 69 million people in this category, constituting 1.1% of the world's population. Projections show that by 2050 this percentage will nearly quadruple, to 4%: there will be 379 million people over the age of 80. The oldest region of the world today is Europe, with a median age of 37.5; this is projected to rise to 49.5 by 2050 (United Nations 2001).

The aging of the world's population poses a challenge and an opportunity to those of us who design technology. Older adults face a range of challenges: physical, social, emotional, and cognitive. It is important to remember that there is not simply a growing absolute number of older adults, but that older adults will constitute an increasingly large fraction of the population. Thus, while it might be desirable to help older adults meet their challenges by providing them with human assistance, the reality is that there are not and will not be enough younger people to provide all the support and assistance needed. An important question then, is how assistive technology can supplement human caregivers to further enhance the lives of older adults.

Many types of assistive technology have been developed. Devices ranging from the relatively commonplace, e.g., better hearing aids, to the futuristic, e.g., intelligent wheelchairs (Yanco 1998), can help older individuals meet physical challenges. Older adults can be supported socially and emotionally through technology that helps alleviate the isolation that is often a problem for them. For example, elder-friendly email systems (Burd ND)and projects such as the the Digital Family Portrait (Mynatt *et al.* 2001), or the Dude's Magic box (Rowan & Mynatt ND) facilitate increased interaction between an older person and his or her family members and friends. This paper focuses on technology that can help older adults meet cognitive challenges they may face. Specifically, it describes the use of automated planning technology to develop *cognitive orthotics*.

The next section provides a brief discussion of one type of cognitive decline that may occur with aging–a decay in prospective memory–and discusses the limitations of many existing cognitive orthotic systems. Following that, the paper introduces Autominder, a cognitive orthotic designed and built at the University of Michigan using planning and plan management techniques. A description of Autominder's architecture is followed by a focused discussion of one of its three main components: the plan manager. Only brief descriptions of the other main components are given, because other papers in this proceedings provide more details of them (Colbry, Peintner, & Pollack 2002; McCarthy & Pollack 2002). The paper concludes by discussing other recent work on developing intelligent cognitive orthotics, and then summarizing the current state of Autominder and our plans for continued work.

Cognitive Orthotics

Cogntive functioning frequently changes with age: just as the body ages, so does the mind (Stern & Carstensen 2001). Cognitive changes may be due to normal aging, or may be the result of diseases that occur with greater frequency in older people. One of the most common causes of severe cognitive impairment, Alzheimer's Disease (A.D.), is strongly correlated with age: approximately 10% of people age 65 and older suffer from A.D., while 20% of those aged 70-84, and nearly 50% of those over 85 have A.D. (AoA 2000). However, at least as important are milder forms of cognitive impairment that may be prior to and often distinct from A.D. The Autominder system described in this paper is aimed primarily at people with mild to moderate cognitive impairment.

One effect of age-related cognitive decline may be decreased prospective memory, leading to forgetfulness about routine daily activities, which the disability-research community call Activities of Daily Living (ADLs) and Instrumental Activities of Daily Living (IADLs). ADLs include fundamental tasks such as eating, drinking, bathing, and toileting, while IADLs include tasks such as managing medicines, managing money, light housekeeping, arranging transportation, preparing meals, and so on. Of course, older individuals may have physical difficulties that impede their ability to perform ADLs and IADLs, but the technology described in this paper is aimed people whose primary impairments are cognitive ones, which prevent them from remembering to perform these activities.[1]

When an older adult no longer consistently performs ADLs and IADLs, he or she may not be able to remain at home, but may need to move either to the home of a relative or to a facility-based setting such as an assisted care

[1]The Autominder system is currently deployed on a mobile robot, and in the future it may be possible to piggyback on the robot other functions that are intended to help meet physical challenges. For instance, the robot could serve as a delivery system: fetching medicine, water, eyeglasses, mail, and so on.

home. It is generally accepted that for many people, postponing such a move as long as feasible is desirable, because people frequently report a better quality of life while they remain in their own homes. Additionally, institutionalization has an enormous financial cost, which must be born by the individual, his or her family, and/or the government.

A number of cogntive orthotics have been proposed over the years to help older adults adapt to cognitive declines and continue satisfactory performance of routine activities. Not all cognitive orthotics have been specifically targeted to older individuals; some have instead been aimed at people with cognitive impairments resulting from other causes, e.g., brain damage resulting from stroke or injury. The idea of using computer technology to enhance the performance of cognitively disabled people dates back nearly forty years (Englebart 1963). Early aids included talking clocks, calendar systems, and similar devices that were not very technologically sophisticated; yet many are still in use today. More recent efforts at designing cognitive orthotics have enabled reminders to be provided using the telephone (Friedman 1998), personal digital assistants (Dowds & Robinson 1996; Jonsson & Svensk 1995) and pagers (Hersh & Treadgold 1994). Research has also aimed at improved modeling of clients' activities, notably in the work of Kirsch and Levine (Kirsch *et al.* 1987), and in the PEAT system (Levinson 1997). However, with the exception of PEAT, which is discussed further in the Related Research section of this paper, these systems generally function in a manner similar to alarm clocks: they provide alarms for prescribed activities at fixed times that are specified in advance by a client and/or his or her caregiver. For example, the web page for a typical cognitive orthotic, the "Schedule Assistant," developed and marketed by AbleLink Technologies, describes its capabilities as follows:

> To set up an appointment or reminder in Schedule Assistant, caregivers use a wizard approach to complete the process of recording a message or reminder, selecting a picture prompt to accompany the message if desired, and setting the time and day for it to play. The system is then able to "wake itself up" to play the appointment message at the desired time(AbleLinkTech 2002).

Although significant attention has been given to the critical issues of usability and interface design in existing systems, less emphasis has been paid to the process of carefully modeling the client's plans, attending to and reasoning about their execution, and deciding whether and when it is most appropriate to issue reminders. Such reasoning is the focus of the Autominder system, described in the next section.

Autominder

The Autominder cognitive orthotic is being developed as part of the Initiative on Personal Robotic Assistants for the Elderly, a multi-university, multi-disciplinary research effort conceived in 1998.[2] The initial focus of the Initiative

[2]In addition to the University of Michigan, the initiative includes researchers at the University of Pittsburgh and Carnegie Mellon University.

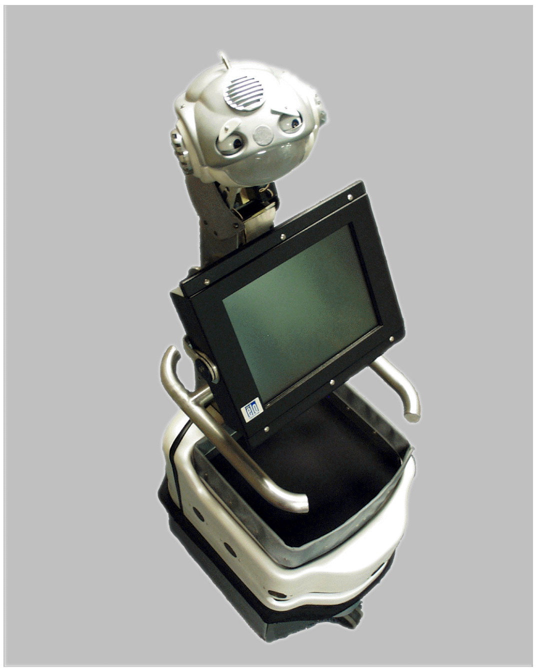


Figure 4: Pearl: A Mobile Robot Platform for the Autominder Cognitive Orthotic. Photo courtesy of Carnegie Mellon University.

is to design an autonomous mobile robot that can "live" in the home of an older individual, and provide him or her with reminders about daily plans. To date, two prototype robots have been designed and built by members of the initiative at Carnegie Mellon. The more recent of these robots, named Pearl, is depicted in Figure 4. Pearl is built on a Nomadic Technologies Scout II robot, with a custom-designed and manufactured "head", and includes a differential drive system, two on-board Pentium PCs, wireless Ethernet, SICK laser range finders, sonar sensors, microphones for speech recognition, speakers for speech synthesis, touch-sensitive graphical displays, and stereo camera systems (Baltus *et al.* 2000; Montemerlo *et al.* 2002; Pineau & Thrun 2002). Members of the Initiative also have interests both in other ways in which mobile robots can assist older people (e.g., telemedicine, data collection and surveillance, and physically guiding people through their environments), and in other platforms for the cognitive orthotic system (e.g., wearable devices and aware homes).

One of the main software components of Pearl is the cognitive orthotic system Autominder, which is being developed by members of the initiative at the University of Michigan. Our goal is to develop a system that is flexible, adaptive, and responsive–and is thus more effective than a glorified alarm clock. To attain this goal, Autominder must maintain an accurate model of the client's daily plan, monitor its performance, and plan reminders accordingly. Consider, for instance, a forgetful, elderly person with urinary incontinence who is supposed to be reminded to use the toilet every three hours, and whose next reminder is scheduled for

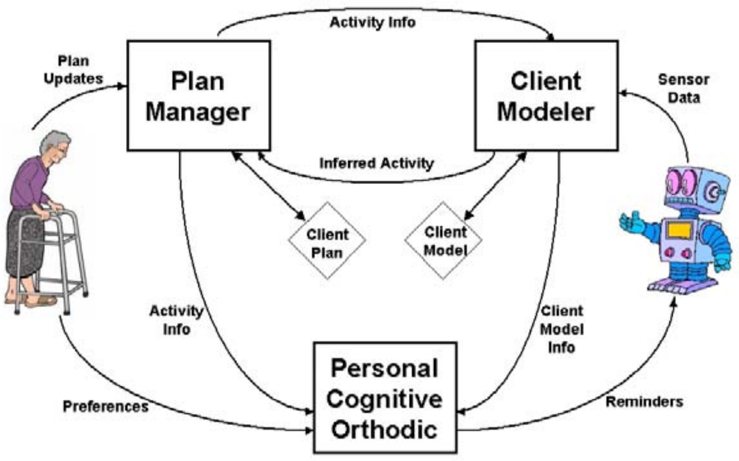


Figure 5: Autominder Architecture

11:00. Suppose that, using its on-board sensors, our robot Pearl observes the person enter the bathroom at 10:40, and conveys this information to Autominder, which concludes that toileting has occurred. In this case, a reminder should not be issued at 11:00, as previously planned. Instead, the client's plan must be adjusted, so that the next scheduled toileting occurs approximately three hours later, i.e., around 13:40. Flexibility is again essential, because a strict three-hour interval may not be optimal. For instance if the client's favorite television program is aired from 13:30 to 14:00, it might be better to issue the reminder at 13:25, and provide a justfication that mentions the television program (e.g., "Mrs. Smith, Why don't you use the toilet now? That way I won't interrupt you during your show.")

Autominder's architecture is depicted in Figure 5. As shown, Autominder has three main components: a Plan Manager (PM), which stores the client's plan of daily activities in the *Client Plan*, and is responsible for updating it and identifying any potential conflicts in it; a Client Modeler (CM), which uses information about the client's observable activities to track the execution of the plan, storing its beliefs about the execution status in the *Client Model*; and a Personal Cognitive Orthotic (PCO), which reasons about any disparities between what the client is supposed to do and what he or she is doing, and makes decisions about when to issue reminders.

Plan Management in Autominder

In Autominder, as in most automated planning systems, we model plans as 4-tuples, $< S, O, L, B >$, where S are steps in the plans, and $O, L,$ and B are temporal ordering constraints, causal links, and binding constraints over those steps.[3] For this application, temporal constraints are very important, and a rich class of such constraints much be supported; specifically, we use the language of disjunctive temporal problems (DTPs) (Oddi & Cesta 2000; Stergiou & Koubarakis 2000; Tsamardinos 2001; Tsamardinos & Pollack 2002) which allows for both quantitative (metric) and qualitative (ordering) constraints, as well as conjunctive and disjunctive combinations of these. We have also recently developed an approach to handling conditional constraints (Tsamardinos, Vidal, & Pollack 2002), but we have not yet implemented these in the Autominder PM.

Formally, each ordering constraint has the form

$$lb_1 \leq X_1 - Y_1 \leq ub_1 \vee \ldots \vee lb_n \leq X_n - Y_n \leq ub_n$$

where the X_i and Y_i refer to the start or end points of steps in the plan, and the lower and upper bounds (lb_i and ub_i) are real numbers. (Without loss of generality, we will assume in this paper that they are integers.) Figure 6 shows how such constraints can be used to express the time at which a step starts or ends, the duration of a step, the amount of time between steps, and so on, as well as expressing ranges and/or disjunctions over such values. Throughout this paper, the start of a step A will be denoted A_S and its end will be denoted A_E. Note that to express a clock-time constraint, e.g., TV watching beginning at 18:00, we use a *temporal reference point* (TR), a distinguished value representing some fixed clock time. In the figure, as well as in the Autominder system itself, the TR corresponds to midnight; the schedule is updated each day.

Note also how the disjunctive constraints can be used to express the fact that two steps cannot overlap. We illustrate this further in Figure 7, which shows a DTP network representing the temporal constraints for a very small plan. The nodes in the network represent the start and end points of each step in the plan, plus the temporal reference point, while the arcs represent the nondisjunctive constraints. The one disjunctive constraint is used to enforce the fact that the two steps in the plan cannot overlap. It should be clear from this example that disjunctive constraints also can be used to express alternative temporal means of resolving a conflict in a plan, i.e., we can represent the possibility of promotion *or* demotion in one constraint.

Plan Initialization

The PM in Autominder is initialized in advance of its use with a specification of the client's daily plan, which is constructed by the client's caregiver, possibly in consultation with the client him- or herself. Different daily plans might be constructed, e.g., one for weekdays and one for weekends, with the appropriate plan loaded each morning, but here we will assume that there is just one daily plan.

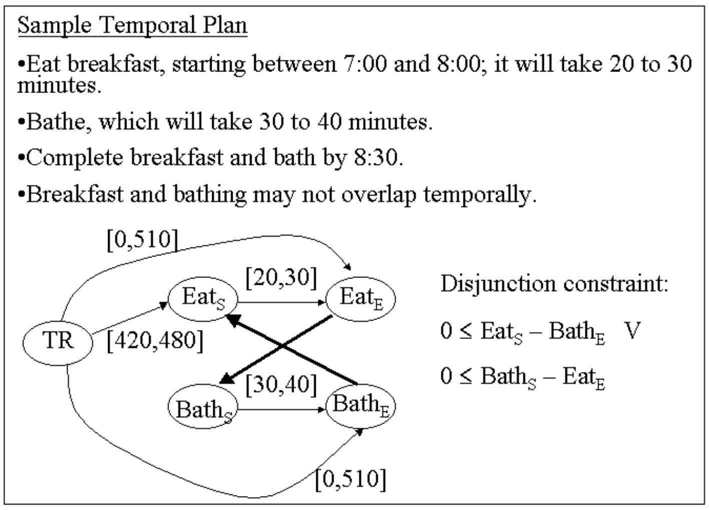
We currently have a rather minimal GUI for specifying a daily plan.[4] It allows one to select pre-constructed plan fragments for routine activities from a library, and to then input specific temporal constraints on the steps in the selected fragments. Thus, a caregiver might begin construction of a typical daily plan by performing the following steps:

[3]In the current version of Autominder, we work with a propositional representation, and thus omit binding constraints. On the other hand, we have an extended class of links allowed: in addition to traditional causal links, we also have implemented inconditions and (simple) resource constraints.

[4]The same GUI can be used for modifying the plan once execution has begun.

"Toileting should begin between 11:00 and 11:15." $660 \leq Toileting_S - TR \leq 675$
"Toileting takes between 1 and 3 minutes." $1 \leq Toileting_E - Toileting_S \leq 3$
"Watching the TV news can begin at 18:00 or 23:00." $1800 \leq WatchNews_S - TR \leq 1802 \vee$ $2300 \leq WatchNews_S - TR \leq 2302$
"The news takes exactly 30 minutes." $30 \leq WatchNews_E - WatchNews_S \leq 30$
"Medicine should be taken within 1 hour of finishing breakfast." $0 \leq TakeMeds_S - EatBreakfast_E \leq 60$
"Toileting and watching the news cannot overlap." $0 \leq WatchNews_S - Toileting_E \leq \infty \vee$ $0 \leq Toileting_S - WatchNews_E \leq \infty$

Figure 6: Examples of the use of DTP Constraints


Sample Temporal Plan

•Eat breakfast, starting between 7:00 and 8:00; it will take 20 to 30 minutes.

•Bathe, which will take 30 to 40 minutes.

•Complete breakfast and bath by 8:30.

•Breakfast and bathing may not overlap temporally.

Figure 7: Temporal Network for a Sample Plan. Note the disjunctive constraint that blocks the steps from overlapping.

- Select a pre-constructed plan fragment for breakfast, which includes three steps–going to the kitchen, making breakfast, and eating breakfast–as well as temporal constraints that order these, causal links that capture their dependencies, and some default durations, e.g., that the eating step will take between 20 and 30 minutes.
- Specify that the first step in the breakfast plan must begin by 7:00, and that the last step must be done by 8:30.
- Select a pre-constructed plan fragment for taking medicine, which we will suppose has only one step–take the medicine–with a default duration of 1 minute.
- Specify an interstep constraint to ensure that the medicine taking occurs ate least two hours after finishing breakfast.

As each pre-constructed plan fragment or constraint is added, the PM performs step merging (Tsamardinos, Pollack, & Horty 2000; Yang 1997), that is, it checks to ensure the consistency of the daily plan being constructed and resolves any conflicts. To do this, it uses the same techniques for consistency checking that are used during plan execution; these techniques are described in the next subsection.

Although our current interface is sufficient for development and testing purposes, it seems clear that further work is required to develop more user-friendly interfaces to allow caregivers to specify plans. Little work has been done on this topic, but see (Miksch *et al.* 1998) for one example of the kinds of interfaces that might be developed.

It is worth stressing that the PM is not a traditional plan-generation system. For the kinds of routine activities that we need to represent in our cognitive orthotic, there seems to be little need to perform planning from scratch. Instead, it is sufficient and more efficient to construct generic plan fragments, and allow the PM to merge these fragments, a process that involves adding new constraints, but not new steps or causal links. In future versions of the system, we may extend the PM to do full-fledged planning or replanning when necessary.

Plan Update

The primary role of the PM is to update the client's plan as the day progresses, ensuring its continued consistency. Update occurs in response to four types of events:

1. *The addition of a new activity to the plan.* The daily plan created at initialization provides a starting point for daily activities, but during the course of the day, the client and/or his or her caregivers may want to make additions to the plan: for instance, to attend a bridge game or a newly scheduled doctor's appointment. At this point, plan merging must be performed to ensure that the overall plan remains consistent. Suppose that the client plan initally specifies taking medicine sometime between 14:00 and 15:00, and that the client then adds a bridge game outside the apartment, to begin at 14:30. The PM must update the plan so that the medicine-taking step precedes the client leaving for the bridge game. (We assume that the medicine must be taken at home.) If, in addition, the medicine-taking must occur at least two hours after each meal, the added restriction on when the medicine will be

taken may also further restrict the time at which lunch should be eaten.

2. *The modification or deletion of an activity in the plan.* This is similar to the previous case: the bridge game might be cancelled, or the doctor's office may change the time of the appointment.[5] The types of required changes are like those needed when an activity is added. Note that the PM will add or tighten constraints if needed, but will not "roll back" (i.e., weaken) any constraints. Continuing the example above, if the bridge game were cancelled, the constraint that the medicine be taken between 14:00 and 14:30 would remain in the plan. More sophisticated plan retraction is an area of future research.

3. *The execution of an activity in the plan.* The PM interacts with another component of Autominder, the Client Modeler (CM). The CM is tasked with monitoring plan execution. It receives reports of the robot's sensor readings, for instance when the client moves from one room to another, and uses that to infer the probability that particular steps in the client plan have been executed; it can also issue questions to the client for confirmation about whether a step has been executed. When the CM believes with probability exceeding some threshold that a given step has begun or ended, it passes this information on to the PM. The PM can then update the client plan accordingly. Suppose again that medicine-taking is supposed to occur at least two hours after the completion of each meal. Upon learning that breakfast has been completed at 7:45, the PM can establish an earliest start time of 9:45 for taking the medicine.

4. *The passage of a time boundary in the plan.* Just as the execution of a plan step may necessitate plan update, so may the non-execution of a plan step. As a very simple example, suppose that the client wants to watch the news on television each day, either from 18:00-18:30 or from 23:00-23:30 p.m. At 18:00 (or a few minutes after), if the client has not begun watching the news, then the PM should update the plan to ensure that the 23:00-23:30 slot is reserved for that purpose. (To keep the example simple, assume that the client always wants to watch from the very beginning of the show.)

To perform plan update in each of these cases, the PM formulates and solves a disjunctive temporal problem (DTP). A DTP is a constraint-satisfaction problem $< V, C >$ where the constrained variables V represent time points–in this case, points corresponding to the start and end of steps–and the constraints C are DTP-constraints, as defined earlier (i.e., disjunctions over differences between time points). The domains for the constrained variables are integers, which in Autominder represent the distance in minutes of the time points from the temporal reference point. For example, a time of 480 might be assigned to the time point that represents the beginning of breakfast; this would correspond

[5]Currently, we allow arbitrary changes to be made to the plan. In subsequent versions of the system, we will need to implement security mechanisms that, for instance, allow the user to make changes to social engagements but not the medicine-taking actions.

```
**Update-Plan-for-Addition**($existing$,$newfrag$)
    E = Convert-to-DTP($existing$)
    N = Convert-to-DTP($newfrag$)
    C = Identify-conflicts($existing \cup newfrag$)
    R = $\emptyset$
    For each member $c$ of C
        R = R $\cup$ a DTP-constraint representing
        the alternative temporal resolutions of $c$
    P = E $\cup$ N $\cup$ R
    P' = Solve-DTP(P)
    Return(Convert-to-Plan-Representation(P'))
```

Figure 8: Algorithm for Update after a Plan Addition

to 8:00 (480 minutes after the temporal reference point of midnight). In fact, we do not need to assign exact times to most time points; instead we find solutions that correspond to maximum allowable time intervals.

To see how this works, consider first the case of updating the plan in response to a plan addition. Psuedo-code for this case is given in Figure 8. The PM begins with the contents of the Client Plan, $existing$, and a plan fragment representing the new activities to be added to the plan, $newfrag$. Both $existing$ and $newfrag$ are encoded as $< S, O, L, B >$ 4-tuples, and so the first step is to convert them to disjunctive temporal problems, E and N, respectively. This is a trivial process that is linear in the number of steps: it involves simply extracting all the temporal constraints and encoding them in a format that our DTP solving engine can handle. Note that there is information lost in the DTP encoding: specifically, the DTP does not encode causal links. Thus, it is crucial that a temporal constraint be explicitly included for each causal link. Additionally, it is necessary to identify all the threats in the union of $existing$ and $newfrag$, a process that is quadratic in the total number of steps. For each identified threat, the PM then constructs a DTP constraint that represents the alternative methods of resolution; call the set of such threat-resolution constraints R. Finally, a plan P that consists of the union of E, N and R is passed to a DTP-solver, which checks for consistency, and returns P augmented by a set of additional constraints that ensure consistency. In particular, if there are any threats in the plan, a resolution will be selected for each one. The last step in the process is to convert the new set of DTP constraints back to a plan tuple.

DTP solving, which is NP-complete, is the only computationally expensive step in the process. In Autominder, we use the Epilitis DTP-solver (Tsamardinos 2001; Tsamardinos & Pollack 2002). Epilitis integrates a number of efficiency heuristics, and has been demonstrated to solve benchmark problems two orders of magnitude faster than the previous state-of-the art solvers. For our current Autominder scenarios, which typically involve about 30 actions, Epilitis nearly always produces solutions in less than one second, a time that is well within the bounds we require.

Like prior DTP solvers (Oddi & Cesta 2000; Stergiou & Koubarakis 2000; Armando, Castellini, & Giunchiglia

```
Update-Plan-for-Modification($existing, mods$)
    $plan$ = Make the modifications in $mods$ to $existing$
        (i.e., remove and/or replace constraints)
    M = Convert-to-DTP($plan$)
    C = Identify-conflicts($plan$)
    R = $\emptyset$
    For each member $c$ of C
        R = R $\cup$ a DTP-constraint representing
        the alternative temporal resolutions of $c$
    P = M $\cup$ R
    P' = Solve-DTP(P)
    Return(Convert-to-Plan-Representation(P'))
```

Figure 9: Algorithm for Update after a Plan Modification

1999), Epilitis does not attempt to solve the DTP directly by searching for an assignment of integers to the time points. Instead, it solves a meta-CSP problem: it attempts to find one disjunct from each disjunctive constraint such that the set of all selected disjuncts forms a consistent Simple Temporal Problem (STP) (Dechter, Meiri, & Pearl 1991). An STP is like a DTP, except that the constraints must be atomic inequalities; no disjunctions are allowed. The details are beyond the scope of the current paper (but see (Tsamardinos 2001; Tsamardinos & Pollack 2002)). The important point here is that by using this approach, Epilitis can return an entire STP, which provides interval rather than exact constraints on the time points in the plan. Consider again our example of the plan that involves taking medicine between 14:00 and 15:00, which is amended with a plan to leave for a bridge game at 14:30. Epilitis will return a DTP that constrains the the medicine to be taken sometime between 14:00 and 14:30; it does not have to assign a specific time (e.g., 14:10) to that action.

The other three cases of plan update are similar. In response to a plan modification, the PM again begins with the current contents of the Client Plan, $existing$, but this time, instead of a second plan to merge in, it has a set of constraints from $existing$ that are to be removed or changed. Thus, it makes the specified modifications to $existing$ and then converts it to a DTP, identifies conflicts, and performs DTP solving as before. The pseudo-code for this is shown in Figure 9.

The psuedo-code for the other two cases of plan update is not shown, as they are similar to the previous ones. In the third case of update, a step S has begun or finished execution. In response, the PM shrinks the temporal constraint(s) associated with the start end, and/or duration of S to a unit interval. For instance, if we know that breakfast began at time 480, then the constraint associated with it becomes $480 \leq EatBreakfast_S - TR \leq 480$. As long as execution has occurred within the legal bounds, there is no need to identify conflicts; instead, the resulting plan with the reduced constraints is passed directly to the DTP solver so that the new tighter constraints can be propagated.

In the fourth case, a time boundary has passed without a step having begun or ended. At this point, the PM must remove the now invalidated disjunct from a constraint, and then attempt to solve the DTP anew. In our TV news example, the plan would include a constraint
$1800 \leq WatchNews_S - TR \leq 1802 \vee$
$2300 \leq WatchNews_S - TR \leq 2302$
i.e., that watching the news must start either right about 18:00, or else about 23:00. If this step has not begun by shortly after 18:00, the first disjunct is no longer viable. Thus, the PM must remove it from the representation of the plan, and attempt to resolve the DTP, using the remaining disjunct. In the current example, there is an alternative disjunction to try. Sometimes, though, when an invalidated disjunct is removed, there may not remain any alternatives; in that case an execution failure has occurred. As with other cases of execution failure, e.g., missed deadlines, Autominder would record this fact, making it available to the caregiver if appropriate.

The discussion of passed time boundaries brings to light one point that was passed over earlier. In general, there may be multiple solutions to a DTP, i.e., multiple consistent STPs that can be extracted from the DTP. In the current version of Autominder, the PM arbitrarily selects one of these (the first one it finds). If subsequent execution is not consistent with the STP selected, then the DTP will attempt to find an alternative consistent solution. A more principled approach would select solutions in an order that provides the greatest execution flexibility. For example, the solution that involves watching the 18:00 news leaves open the possibility of instead watching the news at 23:00. If the first solution found instead involved watching the later news show, then after an execution failure there would be no way to recover, as it would be too late to watch the 18:00 news. Unfortunately selecting DTP solutions to maximize flexibility is a difficult problem (Tsamardinos, Pollack, & Ganchev 2001).

Other Autominder Components

In addition to the PM, Autominder has two other principal components. The Client Modeler (CM) was mentioned above in the discussion on updating the plan in response to plan execution. As noted there, the job of the CM is to monitor the execution of the plan, attempting to infer its status from information obtained from the robot sensors and requesting confirmation from the client when appropriate. To build the CM, we have been adapting Bayesian inference mechanisms to handle the temporal demands of this application; details can be found in (Colbry, Peintner, & Pollack 2002).

The remaining component of Autominder is the Personal Cognitive Orthotic (PCO), which is responsible for making the decision about what reminders to issue and when. To do this, the PCO reasons about the client plan and the client model, identifying any evolving discrepancies between them. It turns out to be relatively easy to generate a legal reminder plan–such a plan simply includes a reminder for every planned activity at the earliest possible time of its execution. However, a reminder plan constructed this way is likely to be a rather poor one when judged by the criteria we use in Autominder, namely:

1. ensuring that the client is aware of activities he or she is expected to perform,
2. increasing the likelihood that the client will perform at least the essential activities (such as taking medicine),
3. avoiding annoying the client, and
4. avoiding making the client overly reliant on the system.

While the simple approach would lead to a reminder plan that satisfies the first criteria, it is unlikely to satisfy the third or fourth, and this in turn may have a negative impact on the second criteria. Consequently, we employ the local search techniques of the Planning by Rewriting algorithm (Ambite & Knoblock 2001) to iteratively search for an improved reminder plan; for details of our approach, see (McCarthy & Pollack 2002)

Related Research

Several existing cognitive orthotics systems were mentioned earlier in this paper. The most notable of these from a plan-based perspective is PEAT (Levinson 1997). This was the first, and to the best of our knowledge, the only marketed cognitive orthotic system that relies on automated planning technology. PEAT, which is marketed primarily to patients with traumatic brain injury, is deployed on a handheld device, and provides visible and audible clues about plan execution. Like Autominder, PEAT maintains a detailed model of the client's plan and tracks its execution, propagating temporal constraints when the client inputs information specifying that an action has been performed. Also, upon the addition of a new action, PEAT simulates the plan to uncover any conflicts, using the PROPEL planning and execution system (Levinson 1995) for this purpose. However, PEAT uses a less expressive planning language than Autominder; it does not attempt to infer the plan execution status; and it does not perform principled reasoning about what reminders to issue when, instead automatically providing a reminder for each planned activity.

Within the past year or two, several new projects aimed at designing intelligent cognitive orthotics have begun to emerge. The MAPS project at the University of Colorado is focusing on the HCI issues involved in building a handheld cognitive orthotic (Carmien 2002). The Independent LifeStyle Assistant Project (ILSA) at Honeywell is another recent related effort, which has some aims that overlap with our own (Miller & Riley 2001). Yet another, even newer project is the Assisted Cognition Project at the University of Washington (Kautz *et al.* 2002). While Autominder is being targeted mainly at people with milder forms of cognitive impairment, the Washington project aims at developing a cognitive orthotic system–an *adaptive prompter*–for people with Alzheimer's disease. The system will use ubiquitous sensors to monitor the performance of routine tasks, and provide prompts when a client gets "stuck". For instance, a sensor in the bathroom might notice that a person with A.D. has picked up a toothbrush but then stopped; in response, the adaptive prompter would provide guidance to the person about putting toothpaste on the brush and using it to brush his or her teeth. As can be seen, the adaptive prompter is targeted at people with more severe cognitive decline than what we imagine for a typical Autominder client.

Conclusions

The Autominder system as described in this paper has been fully implemented in Java and Lisp on Wintel platforms; we are also working on a Web-based interface for plan initialization and update. The most recent version system has been tested in the laboratory; an earlier version was integrated with the robot software and included in a preliminary field test conducted at the Longwood Retirement Community in Oakmont, PA in June, 2001. The goals of that test were, first, to ensure that the robot control software and the cognitive orthotic would work together, and second, to get an initial sense of the acceptability of such a system to older individuals. On both accounts, the test was successful. Admittedly, the older adults who enrolled in the studies were volunteers, and people likely to be intimidated or put off by this type of technology would not have volunteered. However, the people who did participate were uniformly excited about the system, as were the staff at Longwood, who made a number of suggestions to us about how this type of technology could also be used to assist them in their caregiving tasks. We intend to conduct interviews later this year with caregivers and residents at Longwood in order to develop more detailed models of the daily plans of several residents, and then to field test a version of Autominder that encodes those plans. These field tests will be more directly focused on the performance of the cognitive orthotics software.

We have a number of plans for the continued development of Autominder, some of which were already mentioned in this paper. We have planned extensions to the individual reasoning modules, for example, adding the ability to handle conditional constraints to the PM; supplementing the PM with full-fledged planning capabilities to support replanning; enabling the CM to learn the patterns of client activity over time, in order to better interpret observed behavior; and developing techniques for providing better justifications for reminders issued by the PCO. We are also interested in the deployment of the system on alternative hardware platforms. Although there are many advantages to using a robot, including the ability to piggyback on other capabilities, there are clearly also reasons to explore handheld and/or wearable devices and ubiqitous sensors to support cognitive orthotics. Finally, after our experiences with the staff at Longwood, we are interested in exploring the use of systems like ours within the facility-based setting. In that context, the system would coordinate the daily plans not only of a single person, but of multiple people, including both the residents and the staff that takes care of them.

Acknowledgements

Primary financial support for this research comes from an ITR grant from the National Science Foundation (IIS-0085796). Some of the supporting technology was developed with funding from the the Air Force Office of Scientific Research (F49620-01-1-0066) and the Defense Advanced Research Projects Agency and Air Force Research

Office, Air Force Materiel Command, USAF (F30602-00-2-0621). The views and conclusions herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFOSR or the U.S. Government.

I would like to thank all the members of the Nursebot team, and especially my students, Laura Brown, Dirk Colbry, Colleen McCarthy, Cheryl Orosz, Bart Peintner, Sailesh Ramakrishnan, and Ioannis Tsamardinos, who do all the hard work, somehow turning my half-baked ideas into theories and working systems.

References

Able Link Tech. 2002. Ablelink technologies, `http://www.ablelinktech.com`. Quotation from product descriptions subpage.

Ambite, J. L., and Knoblock, C. A. 2001. Planning by rewriting. *Journal of Artificial Intelligence Research* 15:207–261.

AoA. 2000. Factsheet on Alzheimer's disease. U.S. Government Administration on Aging, `http://www.aoa.gov/factsheets/alz.html`.

Armando, A.; Castellini, C.; and Giunchiglia, E. 1999. SAT-based procedures for temporal reasoning. In *5th European Conference on Planning*.

Baltus, G.; Fox, D.; Gemperle, F.; Goetz, J.; Hirsch, T.; Margaritis, D.; Montemerlo, M.; Pineau, J.; Roy, N.; Schulte, J.; and Thrun, S. 2000. Towards personal service robots for the elderly. In *Workshop on Interactive Robots and Entertainment*.

Burd, L. N.D. I-mail. `http://www.cs.colorado.edu/~l3d/clever/projects/i_mail/index.html`.

Carmien, S. 2002. MAPS: PDA scaffolding for independence for persons with cognitive impairment. `http://www.cs.colorado.edu/~l3d/clever/projects/maps/`.

Census 2000. Census report 2000: National projections program/National estimates program. Population Division, U.S. Census Bureau, Washington, D.C. 20233.

Colbry, D.; Peintner, B.; and Pollack, M. E. 2002. Execution monitoring with quantitative temporal bayesian networks. In *Proceedings of the 6th International Conference on AI Planning and Scheduling*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Dowds, M. M., and Robinson, K. 1996. Assistive technology for memory impairment: Palmtop computers and TBI. In *Braintree Hospital Traumatic Brain Injury Neurorehabilitation Conference*.

Englebart, D. C. 1963. A conceptual framework for the augmentation of man's intellect. In *Vistas in Information Handling*. Spartan Books. 1–13.

Friedman, R. H. 1998. Automated telephone conversation to assess health behavior and deliver behavioral interventions. *Journal of Medical Systems* 22:95–101.

Hersh, N. A., and Treadgold, L. 1994. Neuropage: The rehabilitation of memory dysfunction by prosthetic memory and cueing. *NeuroRehabilitation* 4:187–197.

Jonsson, B., and Svensk, A. 1995. Isaac: A personal digitial assistant for the differently abled. In *Proceedings of the 2nd TIDE Congress*, 356–361.

Kautz, H.; Fox, D.; Borriello, G.; and Arnstein, L. 2002. An overview of the assisted cognition project. In *AAAI Workshop on Automation as Caregiver*. To appear.

Kirsch, N.; Levine, S. P.; Fallon-Kureger, M.; and Jaros, L. A. 1987. The microcomputer as an 'orthotic' device for patients with cognitive deficits. *Journal of Head Trauma Rehabilitation* 2:77–86.

Levinson, R. 1995. A general programming language for unified planning and control. *Artificial Intelligence* 76.

Levinson, R. 1997. PEAT–the planning and execution assistant and trainer. *Journal of Head Trauma Rehabilitation* 769.

McCarthy, C. E., and Pollack, M. E. 2002. A plan-based personalized cognitive orthotic. In *Proceedings of the 6th International Conference on AI Planning and Scheduling*.

Miksch, S.; Kosara, R.; Shahar, Y.; and Johnson, P. 1998. Asbruview: Visualization of time-oriented, skeletal plans. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems*, 11–18.

Miller, C., and Riley, V. 2001. The independent lifestyle assistant. In *Proceedings of the XVIIth World Congress of the International Association of Gerontology*.

Montemerlo, M.; Pineau, J.; Roy, N.; and Thrun, S. 2002. Experiences with a mobile robotic guide for the elderly. In *18th National Conference on Artificial Intelligence*. To appear.

Mynatt, E. D.; Rowan, J.; Craighill, S.; and Jacobs, A. 2001. Digital family portraits: Providing peace of mind of extended family members. In *Proceedings of the ACM Conference on Human Factors in Comuting Systems*, 333–340.

Oddi, A., and Cesta, A. 2000. Incremental forward checking for the disjunctive temporal problem. In *European Conference on Artificial Intelligence*.

Pineau, J., and Thrun, S. 2002. High-level robot behavior control using POMDPs. In *AAAI-02 Workshop on Cognitive Robotics*. To appear.

Rowan, J., and Mynatt, E. D. N.D. Dude's magic box and grandma's lap desk. `http://www.cc.gatech.edu/fce/ecl/projects/dude/index.html`.

Stergiou, K., and Koubarakis, M. 2000. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence* 120:81–117.

Stern, P. C., and Carstensen, L. L., eds. 2001. *The Aging Mind: Opportunities in Cognitive Research*. Washington, D.C.: National Academy Press.

Tsamardinos, I., and Pollack, M. E. 2002. Efficient solution techniques for disjunctive temporal problems. Under review; available upon request from the authors.

Tsamardinos, I.; Pollack, M. E.; and Ganchev, P. 2001. Flexible dispatch of disjunctive plans. In *Proceedings of the 6th European Conference on Planning*, 417–422.

Tsamardinos, I.; Pollack, M. E.; and Horty, J. F. 2000. Merging plans with quantitative temporal constraints, temporally extended actions, and conditional branches. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling*.

Tsamardinos, I.; Vidal, T.; and Pollack, M. E. 2002. CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints*. To appear.

Tsamardinos, I. 2001. *Constraint-Based Temporal Reasoning Algorithms with Applications to Planning*. Ph.D. Dissertation, University of Pittsburgh Intelligent Systems Program, Pittsburgh, PA.

United Nations. 2001. World population prospects: The 2000 revision highlights. United Nations Population Division. `http://www.un.org/esa/population/publications/wpp2000/wpp2000h.pdf`.

Yanco, H. A. 1998. Wheelesley, a robotic wheelchair system. In Mittal, V. O.; Yanco, H. A.; Aronis, J.; and Simpson, R., eds., *Lecture Notes in Artificial Intelligence: Assistive Technology and Artificial Intelligence*. Springer-Verlag. 256–268.

Yang, Q. 1997. *Intelligent Planning: A Decomposition and Abstraction Based Approach*. New York: Springer.

APPENDIX D

Assessing the Probability of Legal Execution of Plans with Temporal Uncertainty

Ioannis Tsamardinos
Department of Biomedical Informatics
Vanderbilt University
Nashville, TN 37232 USA
ioannis.tsamardinos@vanderbilt.edu

Martha E. Pollack
Computer Science and Engineering
University of Michigan
Ann Arbor, MI 48109 USA
pollackm@eecs.umich.edu

Sailesh Ramakrishnan
QSS Group Inc.
NASA Ames Research Center
Moffett Field, CA 94305 USA
sailesh@email.arc.nasa.gov

Abstract

Temporal uncertainty is a feature of many real-world planning problems. One of the most successful formalisms for dealing with temporal uncertainty is the Simple Temporal Problem with uncertainty (STP-u). A very attractive feature of STP-u's is that one can determine in polynomial time whether a given STP-u is dynamically controllable, i.e., whether there is a guaranteed means of execution such that all the constraints are respected, regardless of the exact timing of the uncertain events. Unfortunately, if the STP-u is not dynamically controllable, limitations of the formalism prevent further reasoning about the probability of legal execution. In this paper, we present an alternative formalism, called Probabilistic Simple Temporal Problems (PSTPs), which generalizes STP-u to allow for such reasoning. We show that while it is difficult to compute the exact probability of legal execution, there are methods for bounding the probability both from above and below, and we sketch alternative candidate algorithms for this purpose. Computing the probability of legal execution allows a temporal planner to decide, when uncertainty is present, whether to accept or reject candidate plans. In addition, lower bound computation has an important side-effect: it provides guidance as to how to execute an STP-u even when it is not dynamically controllable.

Introduction

Many real-world planning problems involve temporal constraints, and a number of planning formalisms and algorithms have been developed to deal with them. One of the most well-known is the Simple Temporal Problem (STP) formalism (Dechter, Meiri, & Pearl 1991), which allows the representation of temporal constraints of the form $X - Y \leq d$, where X and Y are the times of occurrence of two instantaneous events in the plan and d is a real number (or infinity). For example, if X and Y denote the start and end points of a single action, then the constraint specifies that the action takes no more than d time units.

The STP formalism, along with generalizations of it, such as the Disjunctive Temporal Problem (DTP) (Stergiou & Koubarakis 2000; Tsamardinos 2001) have been very fruitful, both for theoretical investigations of temporal planning (Smith, Frank, & Jónsson 2000) and for practical deployment, notably in NASA's Remote Agent (Muscettola *et al.* 1998). However, these formalisms do not allow any explicit representation of uncertainty. Yet in most interesting, real-world domains, there are many types of uncertainty. One type of uncertainty is associated with conditional execution of actions that depend on observations and the status of the world during execution. The Conditional Temporal Problem (Tsamardinos, Vidal, & Pollack 2003) is an extension of the STP that is able to encode and reason with quantitative temporal constraints and conditional branches.

Another type of uncertainty, that this paper addresses, is *temporal uncertainty*, i.e., uncertainty about the time at which particular events will occur. Such events are said to *uncontrollable*, to distinguish them from the events that are under the control of the agent executing the plan. Often, plans must include temporal constraints that involve uncontrollable events: for instance, it may be necessary to respond to an alarm within two minutes of its going off. The time of the alarm is not within the control of the execution agent, but the time of the responsive event is.

In order to model uncontrollable events, an extended formalism, called Simple Temporal Problems with Uncertainty (STP-u) was developed (Vidal & Ghallab 1996; Vidal & Fargier 1997; Morris, Muscettola, & Vidal 2001). With STP-u's, one can model plans that contain constraints involving uncontrollable events. Notice that with such plans, decisions about when to perform actions must often be deferred until execution time. For instance, one cannot decide in advance when to respond to an alarm: all one can do is wait until the alarm goes off and then respond accordingly.

A very attractive feature of STP-u's is that one can determine in polynomial time whether a given STP-u is *dynamically controllable*, i.e., whether there is a guaranteed means of execution such that all the constraints are respected, regardless of the exact timing of the uncertain events[1]. Not all STP-u's are dynamically controllable. As a simple example, consider an STP-u that includes a constraint requiring an agent to respond to an uncontrollable alarm exactly two minutes *before* it goes off. If the agent doesn't control the alarm–does not know when it will go off and does not have any means of making it go off–then clearly the agent cannot act in a way to satisfy this constraint.

A plan generation system can approach the task of plan-

[1]Here, and in the rest of the paper we assume the only source of uncertainty in the plan is the temporal uncertainty of the uncontrollable events.

ning under temporal uncertainty by generating a plan formulated as an STP-u and then checking to see whether it is dynamically controllable. If it is, the planner can declare success. Unfortunately, if it is not, limitations of the STP-u formalism preclude further reasoning both about the probability of legal execution, and about strategies to maximize that probability. Consequently, the planning system does not know whether to adopt the plan or to search for an alternate, and, if the former–if it does adopt the plan–it is unable to formulate a effective means of executing it.

In this paper, we present a new formalism, called Probabilistic Simple Temporal Problems (PSTPs), which generalizes STP-u's in a way that supports reasoning about the probability that a plan with temporal uncertainty will be legally executed. Although is is difficult to compute an exact probability, we show that there are methods for bounding the probability both from above and below. An upper bound can be used to reject a current candidate plan which falls below a given threshold, while a lower bound can be used to accept a candidate plan.

The remainder of our paper is organized as follows. In the Background section we review the background material on STPs, STP-u's, and dynamic controllability, and in the next section we introduce the Probabilistic Simple Temporal Problems (PSTP) formalism. In the following two sections we describe the technique for computing the upper bound and lower bound on the probability of correctly executing a PSTP, respectively. In particular, we explain how the problem of computing the lower bound can be addressed by first converting a PSTP to an STP-u and then tightening the bounds on that STP-u until it becomes dynamically controllable. The following three sections then sketch some approaches to this lower bound computation.

Finally, the Discussion section summarizes the main ideas, open questions, and future directions. We note there in particular that the lower bound computation has an important side-effect: *it results in the specification of an execution strategy that maximizes the probability of satisfying all the execution constraints.* From another perspective this means that it provides guidance as to how to execute an STP-u even when it is not dynamically controllable.

Background

The formalism used to represent temporal information and uncertainty is based on the Simple Temporal Problem (STP) defined below:

Definition 1. Simple Temporal Problem, STP Solution, Consistent STP. A Simple Temporal Problem (**STP**) is a pair $< V, E >$, where

- V is a set of variables (also called nodes, events, or time-points) taking real values representing the time of occurrence of instanteneous events.
- E a set of temporal constraints on the variables of the form $X - Y \leq b$, $X, Y \in V$, and $b \in \Re \cup \{-\infty, \infty\}$.

A **solution** to an STP is an assignment to the variables that satisfies the constraints. An STP is **consistent** if there exists at least one solution.

STPs have been used to represent temporal plans by using a variable for each action's start and end point. For example, if $start(A)$ and $end(A)$ are the events of starting and ending action A, then the constraints $5 \leq end(A) - start(A) \leq 10$ specify the duration of the action to be between 5 and 10 time units.

STP constraints are binary. In order to represent unary constraints on absolute execution time, e.g., $100 \leq start(A) \leq 200$, a special variable called time reference point TR is defined and is assigned time zero; then, the constraint above can be written as $100 \leq start(A) - TR \leq 200$.

By using an all-pairs shortest path algorithm one can discover the distance from Y to X denoted as d_{YX} and defined as follows:

Definition 2. The **distance** from variable Y to variable X denoted as d_{YX} is the smallest number for which the equation $X - Y \leq d_{YX}$ holds in all STP solutions.

STPs can be executed with minimal on-line constraint propagation as discussed in (Muscettola, Morris, & Tsamardinos 1998; Tsamardinos, Morris, & Muscettola 1988). An STP does not represent uncertainty information about the occurence of events. All variables are assumed to be under the direct control of the agent executing the represented plan and so, if there exists a solution, then the STP is executable in a way that satisfies its constraints. To address this representational limitation, the Simple Temporal Problem with Uncertainty (STP-u) formalism was developed (Vidal & Ghallab 1996; Vidal & Fargier 1997; Morris, Muscettola, & Vidal 2001).

An STP-u makes a distinction between controllable and uncontrollable variables. *Controllable* variables are the ones whose timing of execution is under the direct control of the agent. *Uncontrollable* variables are the ones whose timing of execution depends on Nature (i.e., exogenous factors). The only information represented regarding the exact timing of an uncontrollable X is that it will occur sometime within the interval $[l, u]$ after a controllable Y, called the *parent* of X. Thus, the STP-u specifies that Nature will respect the constraint $l \leq X - Y \leq u$. These constraints involving Nature are called *contingent links* and are distinct from the constraints the plan has to respect to be legally executed, called *requirement links.*

Definition 3. Simple Temporal Problem with Uncertainty. A Simple Temporal Problem with Uncertainty **STP-u** is a tuple $< V_C, E, V_U, C >$, where

- V_C and V_U are the set of controllable and uncontrollable variables, respectively, taking real values.
- E is a set of constraints (requirement links) of the form $X - Y \leq b$, $X, Y \in V_C \cup V_U$, and $b \in \Re \cup \{-\infty, \infty\}$.
- C is a set of contingent links of the form $l \leq X - Y \leq u$, $Y \in V_C$, $X \in V_U$, and $l, u \in \Re$.

Figure 4 shows an STP-u with two controllable variables A, B and an uncontrollable variable C. The edge $A \rightarrow B$ in the figure, annotated with the interval $[p, q]$ graphically expresses the constraints $p \leq B - A \leq q$, i.e., $A - B \leq -p$ and $B - A \leq q$. Similar constraints hold for the other edges. The edge $A \rightarrow C$ is a contingent link, i.e., a constraint Nature is expected to observe.

Contingent constraints are always between a controllable variable Y and an uncontrollable variable X. A contingent link $l \leq \mathit{end}(A) - \mathit{start}(A) \leq u$ may be used for example to specify that the expected duration of an action A is between l and u time units; however, this duration is not something that is determined by the agent.

An STP-u, like an STP, should be executed in such a way that all its constraints are satisfied. However, as we illustrated in the introduction with the alarm example, the existence of uncontrollable events means that decisions about the timing of controllable events may need to be deferred to execution time.

Definition 4. Legal Execution, Execution Strategy. A **legal execution** of a STP-u $< V_C, E, V_U, C >$ is a schedule (time assignment) of occurrences of the events (variables) in $V_C \cup V_U$ in a way that satisfies all the constraints in E. An **execution strategy** is an algorithm that decides when to execute the next controllable action given the execution constraints *and* the observed history of the uncontrollable events.

Definition 5. Dynamic Controllability. (Informal) An STP-u $< V_C, E, V_U, C >$ is **dynamically controllable** if there exists an execution strategy that results in a legal execution regardless of when the uncontrollable variables in V_U occur (provided they occur within the bounds specified by the contingent constraints in C).

For a formal definition of dynamic controllability, see (Morris, Muscettola, & Vidal 2001), which also provides a polynomial-time algorithm for checking whether a given STP-u is dynamically controllable.

How does a domain expert model temporal uncertainty when specifying the constraints for a planner? For any temporally uncertain event A, the expert must specify some bounds on the time of occurrence of A. In the STP-u formalism, this corresponds to setting the values of l and u in a contingent link $l \leq \mathit{end}(A) - \mathit{start}(A) \leq u$. The looser these bounds are set to be, the more likely they are to be observed by Nature, and hence, the more accurate the model is; in the extreme case, if they are set to positive and negative infinity, then the expert is certain that the event will occur within the specified bounds. On the other hand, as the bounds get looser, the likelihood decreases that the STP-u is dynamically controllable. And when the STP-u is not dynamically controllable, the formalism provides no guidance about when to perform the controllable events so as to increase the probability of observing the constraints. Currently, there are no principled procedures for deciding the bounds of the uncontrollables in a way that maximizes the probability that Nature will respect them and that the resulting STP-u will be dynamically controllable.

In fact, because STP-u's do not explcitly represent probability distributions of uncontrollable events, they lack the information needed for such decisions. Probabilistic Simple Temporal Problems (PSTPs), first presented in (Tsamardinos 2002), are an extension of STP-u that includes such information in the temporal plan.

Probabilistic Simple Temporal Problems

We begin by defining PSTPs.

Definition 6. A Probabilistic Simple Temporal Problem **PSTP** is a tuple $< V_C, E, V_U, Par, \mathcal{P} >$, where:

- V_C is the set of controllable variables with real values.
- V_U is a set of real random variables (uncontrollable variables).
- E a set of constraints of the form $X - Y \leq b$, $X, Y \in V_C \cup V_U$, and $b \in \Re \cup \{-\infty, \infty\}$.
- Par is a function $V_U \rightarrow V_C$ that specifies for each uncontrollable its controllable parent.
- $\mathcal{P}$ is a set of conditional probability density functions (pdf) $p_X(t)$ for each $X \in V_U$ providing the mass of probability of X occurring t time units *after* $Par(X)$.

It is worth noting that in PSTPs, each uncontrollable event has a single parent, which must be a controllable event; thus, function Par is well-defined. The probability functions in $\mathcal{P}$ deserve further comment. $\mathcal{P}$ is a set of probability distributions $p_X(t)$ summarizing expectations about the occurrence of each uncontrollable event X. More precisely, given $p_X(t)$, the probability of X occurring T time units or less after $Par(X)$ has occurred is given by $P_X(t \leq T) = \int_{-\infty}^{T} p_X(t)dt$.

Implicit in our definitions is that the probability distribution of occurence of X with time does not depend on absolute time but on relative time from the moment the parent of X is executed ($p_X(t)$ is time invariant).

As an example, suppose that we want to model the fact that an action A has duration normally distributed with mean duration of half an hour ($30'$) and $5'$ standard deviation σ. We define the beginning of the action as the controllable $Y = \mathit{start}(A)$, and the end of the action as the uncontrollable $X = \mathit{end}(A)$, for which $p_X(t)$ follows $Normal(30, 5)$ (normal with half an hour mean and 5 minutes standard deviation). Then we can find out the probability that the action will finish within in 40 minutes after Y (i.e., after we started the action): $P(t \leq 40) = \int_{-\infty}^{40} p_X(t)dt = \Phi(\frac{40-30}{5}) = 97.72\%$, where $\Phi(z)$ is the integral of the Normal(0,1) at point z.[2]

Let us compare modelling action A in a PSTP with modelling the same action in an STP-u. In an STP-u one has to come up with reasonable bounds l and u and specify that $l \leq X - Y \leq u$ is a contingent link to be included in the plan. In comparison, in an PSTP the same constraint is represented by specifying that the parent of X is Y and that X will occur t time units after Y where t follows a normal probability distribution with mean $30'$ and standard deviation $5'$.

Given a PSTP, our goal is to assess the probability that all its execution constraints E will be satisfied during execution. More specifically, we would like to calculate *the probability of the plan being legally executed under an optimal execution strategy*. The reason for doing this is to provide

[2]This integral cannot be solved analytically but is typically computed numerically or provided in a table form.

guidance to the planning process: we want to know whether the current plan is "good enough", i.e., likely enough to succeed, or whether, instead, further effort should be put into looking for a better plan.

(Tsamardinos 2002) shows how to find an optimal execution strategy for PSTPs under certain conditions. However, the basic approach there is a static one, i.e., one that corresponds to strong controllability in STP-u's. To compute the equivalent of a dynamic execution strategy, it is necessary to iterate the process of finding an optimal PSTP execution strategy whenever an observation of an uncontrollable occurs. Computing the exact probability of success of this overall dynamic strategy is difficult. However, we do know how to compute bounds on the probability of success, and that is what we focus on in the remainder of this paper. In the next section, we show how to compute an upper bound on the probability of success. This bound can be used by a system to reject a plan, if it is too low. In the following sections, we describe how to compute a lower bound, by converting a PSTP to an STP-u and then tighening the latter's bounds until it becomes dynamically controllable.

Bounding the Probability of Legal Execution from Above

Suppose that an uncontrollable variable X with parent Y occurs t time units after Y, i.e., $X - Y = t$. If there is no solution to the constraints in E that admits a value t for the difference $X - Y$ then the probability of completing the execution in a way that respects the constraints is zero.

As we mentioned earlier, the distance between Y and X is the minimum number d_{YX} for which $X - Y \leq d_{YX}$ holds in all solutions. Similarly, $Y - X \leq d_{XY}$ holds in all solutions. These inequalities together imply that $-d_{XY} \leq X - Y \leq d_{YX}$ in any legal execution. Therefore, with probability $p_X(t)$ for t outside the interval $[-d_{XY}, d_{YX}]$ a legal execution cannot be achieved. Equivalently, a legal execution can be achieved with probability density at most $p_X(t)$ for t within the interval $[-d_{XY}, d_{YX}]$.

Assuming all uncontrollable events are independent of each other, and using $Success$ to denote the event of a legal execution occuring, then:

$$P(Success) \leq \prod_{X \in V_U, Y = Par(x)} P_X(t \in [-d_{XY}, d_{YX}])$$

$$= \prod_{X \in V_U, Y = Par(X)} P_X(-d_{XY} \leq t \leq d_{YX})$$

The distances d_{XY} can be determined with a polynomial all-pairs shortest path algorithm. The calculation of the probabilities in the product depends on the exact density functions. As an example, if $p_X(t)$ is uniform in the interval $[a, b]$, then $P_X(-d_{XY} \leq t \leq d_{YX}) = \frac{d_{XY} - (-d_{YX})}{b-a} = \frac{d_{YX} + d_{XY}}{b-a}$), assuming $[-d_{XY}, d_{YX}] \subseteq [a, b]$.

As an example consider the PSTP in Figure 1. The dotted edges represent temporal constraints. Specifically, each edge $A \rightarrow B$ annotated with the interval $[l, u]$ represents the

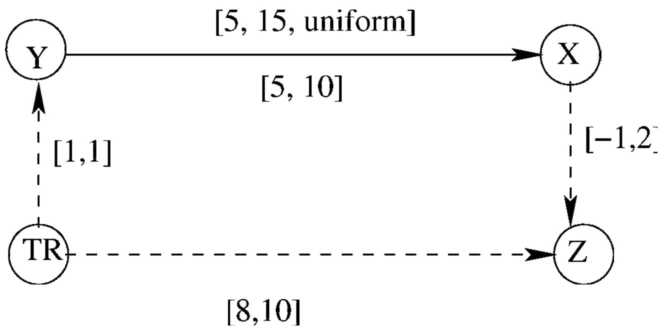


Figure 1: Example for calculating the upper bound.

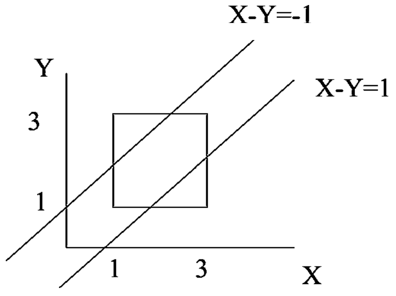


Figure 2: The polytope defined by the constraints provides a tighter upper bound.

two inequality constraints $l \leq B - A \leq u$. There are three such constraints (six single inequality constraints):

$$1 \leq Y - TR \leq 1$$
$$8 \leq Z - TR \leq 10$$
$$-1 \leq Z - X \leq 2$$

We can rewrite these as:

$$-1 \leq TR - Y \leq -1$$
$$8 \leq Z - TR \leq 10$$
$$-2 \leq X - Z \leq 1$$

By adding them up we infer that $5 \leq X - Y \leq 10$. This inferred constraint is depicted in the figure with the annotation $[5, 10]$ below the solid line. It is equivalent to calculating the distances between X and Y: $d_{XY} = -5$ and $d_{YX} = 10$.

The solid link denotes the fact that X is an uncontrollable variable with parent Y and the interval on top of this edge shows the form of this dependency: X will occur some time within the interval $[5, 15]$ after Y has been executed with uniform distribution. For example Y may be the beginning of an action with duration between 5 and 15 and X the end of this action.

As calculated, there is a solution to the constraints only if $5 \leq X - Y \leq 10$. Thus, with at most probability of $P_X(5 \leq t \leq 10) = \frac{10-5}{15-5} = \frac{1}{2}$ the PSTP can be executed successfully.

The upper bound calculated with the above method may not be tight in general. We now discuss ideas why this is the case and how to find tighter bounds. Consider a PSTP with two uncontrollables X and Y that both occur with uniform probability in $[1, 3]$ after the time reference point TR. Let us assume that the only constraints in this PSTP are $-1 \leq X - Y$ and $X - Y \leq 1$.

Figure 2 shows the space of legal executions. The x-axis is the time of occurence of X and similarly for Y. Since $TR = 0$ the set of legal execution is the area of the rectangle bounded by the lines $X - Y = -1$ and $X - Y = 1$. Calculating the bounds with the method that we provided yields: $P_X(-d_{X,TR} \leq t \leq d_{TR,X})P_Y(-d_{Y,TR} \leq t \leq d_{TR,Y}) = P_X(-\infty \leq t \leq \infty)P_Y(-\infty \leq t \leq \infty) = 1$.

A tighter bound on the probability in this example would be the area of the rectangle bounded by the two constraints. In general, a tighter bound could be obtained by calculating the mass of probability contained in the polytope defined by the constraints.

The mass of probability of this polytope is still only an upper bound on the probability of correct execution: even though for every point in this polytope (i.e., execution) the constraints are satisfied, that does not mean that an agent will dynamically be able to construct this solution, unless it is clairvoyant in the general case.

Bounding the Probability of Legal Execution from Below

Let $< V_C, E, V_U, Par, \mathcal{P} >$ be a PSTP and suppose that we are given intervals $[l_X, u_X]$ for each uncontrollable variable X with parent Y. The PSTP and the intervals can be seen as corresponding to an STP-u with the same controllable and uncontrollable variables, same constraints, and contingent links $l_X \leq X - y \leq u_X$ for each uncontrollable variable.

If this STP-u is dynamically controllable then there exists an execution strategy for legally executing the STP-u no matter when the uncontrollables occur *within these intervals*. Thus, in all such cases where the uncontrollables occur within these bounds an agent can execute the plan with probability one, provided it follows the execution strategy returned by the STP-u controllability algorithm. The probability of all such cases is thus a lower bound on the probability of a legal execution. Thus:

$$P(Success) \geq \prod_{X \in V_U} P_X(l_X \leq t \leq u_X)$$

Unlike the upper bound that we provided in the previous section, in the lower bound case the bounding intervals $[l_X, u_X]$ cannot be easily computed (because it is required that the corresponding STP-u is controllable). The looser these intervals the tighter the lower bound will be. To find the tightest bound possible one needs to solve the following optimization problem:

Definition 7. Lower Bound Problem.

Given PSTP $< V_C, E, V_U, Par, \mathcal{P} >$:

Maximize $\prod_{X \in V_U} P_X(l_X \leq t \leq u_X)$
subject to:
$< V_C, E, V_U, C >$ being dynamically controllable
where C is the set of contingent links
$\{l_X \leq X - Y \leq u_X \mid X \in V_U, Y = Par(X)\}$
and decision variables:
l_X, u_X, for each $X \in V_U$

Unfortunately, it is difficult to directly apply typical constraint optimization techniques such as gradient descent or Newton's Method on this problem. This is because such methods require expressing the feasible set as the decision variable vectors that satisfy a set of equality or inequality constraints. In the lower bound problem the feasible region is the set of decision variable vectors that satisfy the single constraint that the corresponding STP-u is dynamically controllable.

In the following two sections we sketch candidate algorithms that approximate the optimal solution to the lower bound problem. We then return to the formulation of the problem as an optimization problem and suggest ways to convert it to a form suitable for classical optimization techniques in a way that approximates the problem we are trying to solve.

Binary Search for Loosest Bounds

In our first algorithm, we perform binary search for the bounds on the uncontrollable intervals that are as loose as possible while still guaranteeing dynamic controllability. The basic algorithm is as follows:

```
1. Given PSTP $< V_C, E, V_U, Par, \mathcal{P} >$, construct a corresponding STP-u $S < V'_C, E', V'_U, C >$ where $V'_C = V_C$, $V'_U = V_U$, $E' = E$, and $C = l_X \leq X - Par(X) \leq u_X$ for each $X \in V_U$, where $l_X$ and $u_X$ are initialized to include most of the mass of probability of $p_X$. (For example, $l_X$ might be the mean minus three standard deviations, while $u_X$ might be the mean plus three standard deviations.)
2. Let $\epsilon$ be a small threshold value, and let $F = 1$.
3. While (S is not dynamically consistent) and ($F > \epsilon$)
4. Begin
5.   If $S$ is not dynamically controllable
6.       $F = F/2$
7.       Reduce all contingent edges in S by a factor of F
8.   Else
9.       $F = 3F/2$
10.      Increase all contingent edges in S by a factor of F
11. End If
12. End While
13. Return S.
```

Note that this algorithm assumes that the underlying STP-u can eventually be made consistent by shrinking the bounds on the uncontrollable events far enough: i.e., if the time points of the uncontrollables could be pinned down, the network would be executable. Also, we have made an arbitary decision about the rate at which we modify the size of the intervals, reducing them by a half when the network is not dynamically controllable, and increasing them by a half when it is. To achieve faster convergence, we may want to vary these values.

This basic algorithm can be improved in several ways. First, when an STP-u is not dynamically uncontrollable, this

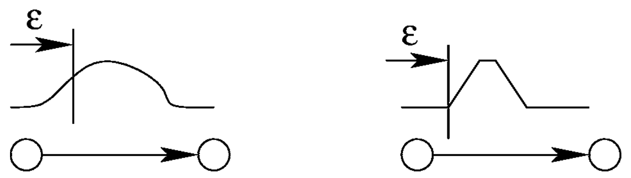


Figure 3: Two uncontrollable events with different distributions.

may be due only to some, and not all, of the uncontrollable events. It may be possible to identify which uncontrollable events are to blame while running the STP-u controllability algorithm and then to modify the above algorithm so that only the edges incident upon culpable events are reduced. Second, the above algorithm does not take account of the fact that the PSTP explicitly models the probability distribution associated with each uncontrollable event. Instead, it reduces the time intervals associated with all events equally. An improved appropriate extension would be to reduce the bounds proportionally to the probability mass associated with each interval. For example, if one contingent interval has a distribution with very wide variance, while another has a much steeper distribution with narrower variance, we would prefer to place tighter bounds on the first–or, put otherwise, shrink the first interval more–than the second, because that would result in less reduction in the overall probability mass of the uncontrollable events modeled. (See Figure 3.)

Iterative Tightening

The Binary Search approach employs the dynamical controllability algorithm as a black box. The Iterative Tightening approach on the other hand modifies the dynamic controllability algorithm.

The Iterative Tightening first converts the PSTP to an STP-u by calculating loose bounds for the uncontrollables in a way that contain most (or all if possible) of the mass of probability (exactly as the Binary Search algorithm). Then, it runs a modified dynamic controllability algorithm: instead of stopping as soon as it is discovered that the STP-u is not controllable, the algorithm relaxes the problem (by tightening the bounds) and continues.

The dynamic controllability algorithm checks each triplet of variables A, B and C, where C is uncontrollable (as shown in Figure 4. The constraints (requirement links) $A \dashrightarrow B$ and $B \dashrightarrow C$ may be explicit or implicit constraints. The algorithm then imposes a set of additional constraints that ensure the existance of an execution strategy. If the propagation of these constraints does not result in a "squeeze" of the contingent link, the STP-u is controllable.

The Iterative Tightening algorithm employs the same strategy. It selects a triangle of variables to work on and imposes the constraints determined by the controllability algorithm. However, if the propagation of these constraints squeeze any other contingent link, instead of stopping, the algorithm tightens the contingent link to these new bounds. Obviously, this algorithm will not return an execution strat-

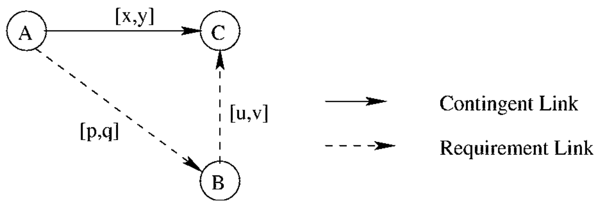


Figure 4: Triangular Networks (Morris, Muscettola, & Vidal 2001).

egy that works for all possible occurences of the uncontrollables of the original bounds, but only for the final tightened bounds.

The algorithm is as follows:

1. Given PSTP $< V_C, E, V_U, Par, \mathcal{P} >$, construct a corresponding STP-u $S < V_C', E', V_U', C >$ where $V_C' = V_C$, $V_U' = V_U$, $E' = E$, and $C = l_X \leq X - Par(X) \leq u_X$ for each $X \in V_U$, where l_X and u_X are initialized to include most of the mass of probability of p_X.
2. Non-deterministically CHOOSE a triangle of variables A, B, C where C is uncontrollable.
3. Impose the constraints determined by the dynamic controllability algorithm.
4. Propagate the constraints as in the controllability algorithm, but allow requirement links to be squeezed.
5. Repeat until the lower bound that corresponds to the current STP-u is high enough, or the last constraint propagation did not change the STP-u.

In Iterative Tightening the order of consideration of each triangle matters. When a triangle A, B, C and a contingent link $A \dashrightarrow C$ is selected and appropriate constraints are imposed to ensure controllability, essentially the algorithm creates an execution strategy that works for all cases where C occurs within the current bounds given for $A \dashrightarrow C$. Propagation of these constraints may require that other uncontrollables occur within a tighter interval to allow for this strategy to work.

For example, suppose that there are two contingent links $Y \dashrightarrow X$ and $A \dashrightarrow B$. Selecting a triangle that involves the first one may cause the bounds on the second one to shrink considerably in order to allow the execution strategy to work with all possible occurences of X. If B's probability distribution has heavy tails, that means that a significant mass or probability will be excluded from the calculation of the lower bound. If instead the second triangle is selected first, its bounds will not be tightened but may cause the bounds on the first link to shrink. If however, the distribution of X has smaller variance, then shrinking the bounds will not exclude as much probability mass and the algorithm will return a tigher lower bound.

Possible variants of the algorithm include a backtracking search where different choices of triangles are made in a search for the STP-u that provides the tighest lower bound on the probability of successful execution.

Non-Linear Optimization Solutions

In this section, we explore the possibility of solving the lower bound problem with optimization methods. While the objective function is suitable for typical optimization methods, the constraints are not. We now attempt to cast the constraint of the resulting STP-u being dynamically controllable, as a set of inequalities, which would allow non-linear optimization techniques to be used.

Consider the triangle of variables and edges of Figure 4, where the edge $A \rightarrow C$ is a contingent link. According to (Morris, Muscettola, & Vidal 2001) the triangle is dynamically controllable if any of the following three conditions hold:

1. $v < 0$, and the triangle is pseudo-controllable.
2. $u \geq 0, B - A \subseteq [y - v, x - u]$, and the triangle is pseudo-controllable.
3. $v \geq 0$, $u < 0$, $y - v \leq x$, and the triangle is pseudo-controllable.

Pseudo-controllability denotes the fact that the bounds $[x, y]$ are not "squeezed" by the contraints of the triangle, or in other words that $[x, y] \subseteq [-d_{CA}, d_{AC}]$. Recall that the dynamic controllability algorithm determines whether there is a way to execute the plan no matter when the uncontrollables occur. The interval $[-d_{CA}, d_{AC}]$ is the interval dictated by the constraints in the plan: any time within that interval participates in at least one solution of the constraints. The interval $[x, y]$ is derived from the contingent link and is a constraint on Nature. Thus, if there are values of $[x, y]$ that do not participate in any solution of the constraints, Nature may select one of these values forbidding the completion of the plan in a way that satisfies the constraints.

Apart from the three cases above for when a triangle is dynamically controllable there is a fourth case. Thus, the above three cases together are sufficient but not necessary conditions. The fourth case involves accepting a ternary and disjunctive constraint that is called a wait, which we will ignore for the moment.

An STP-u is dynamically controllable if all such triangles in the network are dynamically controllable. These three cases direct the design of the our algorithm.

1. Given a PSTP $< V_C, E, V_U, Par, \mathcal{P} >$.
2. Define a non-linear optimization problem with decision variables x_i, y_i for every uncontrollable, objective function $\prod_i P_C(x_i \leq t \leq y_i)$, and inequality constraints S as defined below.
3. Initialize $S \leftarrow E$.
4. For each triple of variables A_i, B_i, C_i as in Figure 4, where C_i is uncontrollable:
   - If $v_i < 0$, then no extra constraint needs to be added.
   - If $u_i \geq 0$, then $S \leftarrow S \cup \{B_i - A_i \subseteq [y_i - v_i, x_i - u_i]\}$
   - Else, $S \leftarrow S \cup \{y_i - v_i \leq x_i\}$.
5. Solve the optimization problem.

The solution to the optimization problem will return a set of values for the decision variables for which all the constraints are satisfied. By construction, satisfying all these constraints implies that the corresponding STP-u is dynamically controllable. This is because each such triangle falls into one of the three cases above.

In addition, in any solution of the optimization problem the triangles are pseudo-controllable. This is because any bounds x, y selected by the optimization for a contingent link $A \rightarrow C$ are as squeezed as possible: $y \leq d_{AC}$ because if $y > d_{AC}$ then y is outside the feasible set imposed by the constraints of the optimization problem.

Intuitively, the algorithm is free to select any x, y bounds on contingent links and impose any constraints on Nature desired. Of course Nature may not observe these constraints but we can calculate the probability that she will and obtain the desired lower bound.

Notice that since the three cases are sufficient but not necessary it is conceivable that this is not the tightest lower bound on the probability that can be achieved using this kind of approach (i.e., by translating to an STP-u). Specifically, there is a fourth case that we omitted from consideration: it involves a disjunctive and ternary constraint called a wait on C. For example $wait < C, 5 >$ means that one should wait to execute B until 5 time units have passed after A has been executed or C has been observed. A non-linear optimization algorithm that takes into consideration this case may be able to further increase the bounds $[x, y]$ to include more mass of probability. It is currently unknown how significant is this case in practice and how much looser than optimal is a lower bound that is achieved by ignoring this case.

We now consider a specific class of probability density functions and the corresponding optimization problems they give rise to.

Optimization for Uniform Distributions

Let us denote with p_i the pdf of the ith uncontrollable variable and suppose that all probability distributions are uniform, that is $p_i(t) = \frac{1}{b_i - a_i}$, when $t \in [a_i, b_i]$ and zero outside this interval.

If $p(t)$ is uniform in $[a, b]$, then

$$P(x \leq t \leq y) = \frac{\min(b, y) - \max(a, x)}{b - a}$$

for $x \leq y$. In Figure 5 this is justified pictorially where the probability density of a uniform p_i within the bounds $[a, b]$ is shown. $P(x \leq t \leq y)$ is the area above the intersection of $[a, b]$ and $[x, y]$.

Instead of maximizing the actual probability of successful execution, we can maximize its logarithm.

$$\max \log \prod_i P_i(x_i \leq t \leq y_i)$$

which is equal to

$$\max \sum_i \log P_i(x_i \leq t \leq y_i)$$

By utilizing the fact that P_i's are uniform, this is equivalent to

$$\max \sum_i \log \frac{\xi_i - \sigma_i}{b_i - a_i},$$

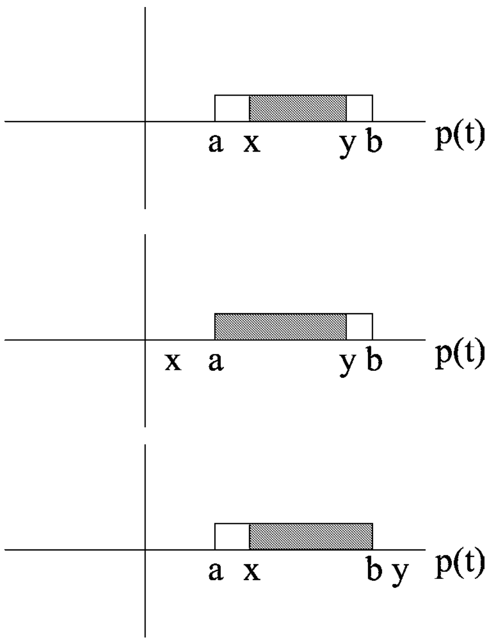


Figure 5: The mass of probability of a random variable uniformly distributed within $[x, y]$ occuring within $[a, b]$ is the area above the intersection of the intervals.

where $\sigma_i = \max(x_i, a)$ and $\xi_i = \min(y_i, b)$. In turn, this gives:

$$\max(\sum_i \log(\xi_i - \sigma_i) - \sum_i \log(b_i - a_i))$$

The last sum is a constant term and can be dropped from the objective function during optimization (but is required to compute the final bound on the probability). So the objective function becomes

$$\max \sum_i \log(\xi_i - \sigma_i),$$

or equivalently

$$\min \; - \sum_i \log(\xi_i - \sigma_i)$$

The constraints of this optimization problem are given by Steps 3 and 4 in the algorithm of the previous section. That is, they are the difference constraints among the PSTP variables (controllable and uncontrollable ones) union the constraints required to guarantee the resulting STP-u is dynamically controllable. In addition to these constraints however, we need to add that $\sigma_i = \max(x_i, a_i), \xi_i = \min(y_i, b_i)$, and that $x_i \leq y_i$.

The max and min functions present problems to most optimization algorithms. Fortunately, in this case we can substitute $\xi_i = \min(y_i, b_i)$ with the constraints $\xi_i \leq b_i, \xi_i \leq y_i$, and $\sigma_i = \max(x_i, a_i)$ with $\sigma_i \geq a_i$, and $\sigma_i \geq x_i$. This is because, in order to maximize the objective function the ξ_i should be as large as possible; so the optimization engine will increase the ξ_i until at least one of the equalities $\xi_i = b_i, \xi_i = y_i$ holds, in which case $\xi_i = \min(y_i, b)$. A similar argument holds for σ_i.

The feasible region defined by these inequality constraints is convex. Additionally, the objective function is twice differentiable everywhere except from the boundary where $\sigma_i = \xi_i$. So the objective function is twice differentiable in the interior of the feasible region.

Let us calculate the second derivative of each term in the sum. Define $f(\xi, \sigma) = -\log(\xi - \sigma)$. Then, $\nabla f(\xi, \sigma) = [-(\xi - \sigma)^{-1}, (\xi - \sigma)^{-1}] = [-a, a]$, for $a = (\xi - \sigma)^{-1}$. The Hessian is $H = \nabla^2 f(\xi, \sigma) = \begin{bmatrix} a^2 & -a^2 \\ -a^2 & a^2 \end{bmatrix}$. H is semidefinite positive because the eigenvalues are non-negative. The eigenvalues λ solve $\det(\lambda I - \nabla^2 f(\xi, \sigma)) = \lambda^2 - 2a^2\lambda = 0$, i.e., $\lambda = 0$ or $\lambda = 2a^2 > 0$ for $\sigma < \xi$. Thus, function f defined on a convex set (when $\sigma \leq \xi$) has a positive semidefinite $\nabla^2 f$ in the interior and thus is convex (provided the interior is non-empty). The objective function, as a sum of such convex functions is also convex.

Convex optimization problems have a unique minimum and in general, are relatively easily solved with modern optimization software. We are currently considering other families of probability distributions such as exponential or normal distributions.

Discussion

The recent literature on planning has shown a growing interest in handling more and more realistic problems, and along with that has come a concern with various types of uncertainty. In this paper, our focus has been on temporal uncertainty: uncertainty about the time at which certain exogenous, or "uncontrollable" events will occur. Significantly, domain experts typically know more about such events than just the interval of time during which they will occur–they often know a probability distribution over the interval of occurrence. Yet the most successful formalism for planning with temporal uncertainty, the STP-u's, don't allow one to exploit that knowledge. Instead, the domain expert must specify fixed bounds on the time during which each uncontrollable event must happen. If he sets the bounds too narrowly, he may produce a plan that is dynamically controllable, but that nonetheless fails, because the uncontrollable event occurs outside the modeled time. If he sets them too widely, he may produce a plan that is not dynamically controllable. And execution strategies only exist for dynamically controllable plans; if an STP-u is not dynamically controllable, there is no effective means of deciding when to execute the controllable actions in it.

This poses a real problem for the designer of a planner dealing with temporal uncertainty. It is difficult to know how to set the bounds on the uncontrollable events; and if the bounds are set too widely, it is impossible to assess the probability that the plan can nonetheless be legally executed, and thus, impossible to make a principled decision about whether to adopt this plan or whether to search further for an alternative.

What we would like to do is to enable the domain expert to specify bounds on the uncontrollable events that are "as wide as possible": by this we mean that they maximize the probability that the events will in fact occur during the modeled bounds, subject to the constraint that the network is dynamically controllable. When the bounds are set in this fashion, there are two results: first, we have an evaluation of the probability that the plan can be legally executed, which can be used to decide whether to accept or reject it, and second, if it is accepted, then the network with the bounds thus set can serve as the basis of a legal execution strategy.

Because STP-u's do not include information about the probability distribution of the timing of uncontrollable events, we presented a generalization of them, called Probabilistic Simple Temporal Problems (PSTPs). Unfortunately, given an PSTP, it is difficult to compute an exact probability of legal execution. What we can do, however, is bound the probability, both from above and below. The upper bound simply provides a way of rejecting a plan if it is not below a given threshold. The lower bound is arguably more interesting, as it is not only what gives a way of accepting a plan, when it is above a threshold, but is also what allows one to approximate the widest possible bounds.

We presented three alternative algorithms for approximating the widest possible bounds. The first performs binary search for a value v that represents the minimal proportion by which all the uncontrollable intervals need to be reduced to achieve a dynamic controllability. The second runs the dynamic controllability algorithm with the modification that it does not exit as soon as controllability is deemed impossible. Instead, it shrinks the intervals appropriately, relaxing the initial problem, until controllability is achieved. The third algorithm takes a very different approach, attempting to reduce the problem to one of non-linear optimization. It approximates the set of controllable STP-u's with a set of inequality constraints. The next major step in this work it to implement these three algorithms and conduct both experimental analyses of their performance in terms of computational efficiency and quality of lower bounds returned. Additionally, it will be important to integrate work on temporal uncertainty of the kind described in this paper with work on causal uncertainty, such as that discussed in (Tsamardinos, Vidal, & Pollack 2003).

References

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Morris, P.; Muscettola, N.; and Vidal, T. 2001. Dynamic control of plans with temporal uncertainty. In *Proceedings of the 17th International Joint Conference on A.I. (IJCAI-01)*. Seattle (WA, USA): Morgan Kaufmann, San Francisco, CA.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote agent: To boldly go where no ai system has gone before. *Artificial Intellience* 103:5–47.

Muscettola, N.; Morris, P.; and Tsamardinos, I. 1998. Reformulating temporal plans for efficient execution. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR-98)*, 444–452.

Smith, D.; Frank, J.; and Jónsson, A. 2000. Bridging the gap between planning and scheduling. *Knowledge Engineering Review* 15(1).

Stergiou, K., and Koubarakis, M. 2000. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence* 120:81–117.

Tsamardinos, I.; Morris, P.; and Muscettola, N. 1988. Fast transformation of temporal plans for efficient execution. In *Proceedings of the 15th National Conference on Artificial Intelligence.*

Tsamardinos, I.; Vidal, T.; and Pollack, M. E. 2003. CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints (to appear), Special Issue on Planning* 8(4).

Tsamardinos, I. 2001. *Constrained-Based Temporal Reasoning Algorithms with Applications to Planning.* Ph.D. Dissertation, University of Pittsburgh.

Tsamardinos, I. 2002. A probabilistic approach to robust execution of temporal plans with uncertainty. In *Proc. of the 2nd Greek National Conference on Artificial Intelligence.*

Vidal, T., and Fargier, H. 1997. Contingent durations in temporal CSPs: from consistency to controllabilities. In *Proceedings of the 4th International Workshop on Temporal Representation and Reasoning (TIME-97)*, 78–85.

Vidal, T., and Ghallab, M. 1996. Dealing with uncertain durations in temporal constraint networks dedicated to planning. In *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI-96)*, 48–52.

APPENDIX E

Lecture Notes in Computer Science

Flexible Dispatch of Disjunctive Plans

Ioannis Tsamardinos[1], Martha E. Pollack[2], and Philip Ganchev[1]

[1] Intelligent Systems Program, University of Pittsburgh, Pittsburgh, PA 15260 USA
tsamard@eecs.umich.edu, ganchev@cs.pitt.edu
[2] Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48103 USA
pollackm@eecs.umich.edu

Abstract. Many systems are designed to perform both planning and execution: they include a plan deliberation component to produce plans that are then dispatched to an execution component, or *executive*, which is responsible for the performance of the actions in the plan. When the plans have temporal constraints, dispatch may be non-trivial, and the system may include a distinct *dispatcher*, which is responsible for ensuring that all temporal constraints are satisfied by the executive. Prior work on dispatch has focused on plans that can be expressed as Simple Temporal Problems (STPs). In this paper, we sketch a dispatch algorithm that is applicable to a much broader set of plans, namely those that can be cast as Disjunctive Temporal Problems (DTPs), and we identify four key properties of the algorithm.

1 Introduction

Many systems are designed to perform both planning and execution: they include a plan deliberation component to produce plans that are then dispatched to an execution component, or *executive*, which is responsible for the performance of the actions in the plan. When the plans have temporal constraints, dispatch may be non-trivial, and the system may include a distinct *dispatcher*, which is responsible for ensuring that all temporal constraints are satisfied by the executive. Prior work on plan dispatch [1-3] has focused on plans that can be represented as Simple Temporal Problems (STP) [4]. In this paper, we sketch a dispatch algorithm that is applicable to a much broader set of plans, those that can be cast as Disjunctive Temporal Problems (DTPs), and identify four key properties of the algorithm.

2 Disjunctive Temporal Problems

Definition. A **Disjunctive *Temporal Problem (DTP)*** is a constraint satisfaction problem $<V, C>$, where V is a set of variables (or nodes) whose domains are the real numbers, and C is a set of disjunctive constraints of the form C_i: $l_1 \leq x_1 - y_1 \leq u_1 \vee$

$\ldots \vee\ l_n \leq x_n - y_n \leq u_n$, such that for $1 \leq i \leq n$, x_i and y_i are both members of V, and l_i , u_i are real numbers. An ***exact solution*** to a DTP is an assignment to each variable in V satistying all the constraints in C. If a DTP has at least one exact solution, it is ***consistent***.

A DTP can be seen as encoding a collection of alternative Simple Temporal Problems (STPs). To see this, note that each constraint in a DTP is a disjunction of one or more STP-style inequalities. Let C_{ij} be the *j*-th disjunct of the *i*-the constraint of the DTP. If we select one disjunct C_{ij} from each constraint C_i, then the set of selected disjuncts forms an STP, which we will call a ***component STP*** of a given DTP. It is easy to see that a DTP D is consistent if and only if it contains at least one consistent component STP. Moreover, any solution to a consistent component STP of D is also clearly an exact solution to D itself.

Definition. A(n inexact) ***solution*** to a DTP is a consistent component STP of it. The ***solution set*** for a DTP is the set of all its solutions.

When we speak of a solution to a DTP, we shall mean an inexact solution. Plans can be cast as DTPs by including variables for the start and end points of each action.

3 A Dispatch Example

Consider a very simple example of a plan with three actions, *P*, *Q*, and *R*. (For presentational simplicity, we assume each action is instantaneous and thus represented by a single node). *P* must occur in the interval [5,10] and *Q* in the interval [15,20]; *P* and *Q* must be separated by at least 6 time units; and *R* must be performed either the interval [11,12] or [21,22]. The plan as described can be represented as the following DTP: {C1. $5 \leq P - TR \leq 10 \vee 15 \leq P - TR \leq 20$; C2. $5 \leq Q - TR \leq 10 \vee 15 \leq Q - TR \leq 20$; C3. $6 \leq P - Q \leq \infty \vee 6 \leq Q - P \leq \infty$; C4. $11 \leq R - TR \leq 12 \vee 21 \leq R - TR \leq 22$}. (Note that *TR*, the time reference point, denotes an arbitrary starting point.) This DTP has four (inexact) solutions: { STP_1: c_{11}, c_{22}, c_{32}, c_{41}; STP_2: c_{11}, c_{22}, c_{32}, c_{42}; STP_3: c_{12}, c_{21}, c_{31}, c_{41}; STP_4: c_{12}, c_{21}, c_{31}, c_{42}}.

Definition: An STP variable *x* is ***enabled*** if and only if all the events that are constrained to occur before it have already been executed. A DTP variable *x* is ***enabled*** if and only if it has a consistent component STP in which *x* is enabled.

In STP_1, both *P* and *R* are initially enabled, while in STP_3 and STP_4, *Q* is initially enabled. Hence, all three actions are initially enabled for the DTP. Enablement is a necessary but not sufficient condition for execution: an action must also be *live,* in the sense that the temporal constraints pertaining to its clock time of execution are satisfied. In the current example, none of the actions are initially live. The first action to become live is *P*, at time 5. An action is *live* during its *time window*.

Definition: The ***time window of an STP variable*** *x* is a pair [*l*,*u*] such that $l \leq x - TR \leq u$, and for all *l'*, *u'* such that $l' \leq x - TR \leq u'$, $l' \leq l$ and $u \leq u'$. Given a set of consistent component STPs for a DTP, we will write TW (*x*,*i*) to denote the time window for variable *x* in the i^{th} such STP. The ***upper bound*** of a time window [*l*,*u*] for *x* in STP *i*, written U(*x*,*i*), is *u*. The ***time window of a DTP variable*** *x* is TW (*x*)=$\cup_{i \in S}$ TW (*x*,*i*), where *S* is the solution set of *D*.

The dispatcher can provide information about when actions are enabled and live in an ***Execution Table (ET).*** This is a list of ordered pairs, one for each enabled action. The first element of the entry specifies the action, and the second is a list of the convex intervals in that element's time window. For our example, then, the initial ET would be: {<*P*, {[5,10], [15,20]}>, <*Q*, {[5,10],[15,20]}}>, <*R*, {[11,12],[21,22]}>}. The ET summarizes the information in the solution STPs so that the executive does not have to handle them directly.

The ET provides information about what actions *may* be performed, but it does not provide enough information for the executive to determine what actions *must* be performed. To see this, note that the ET just given does not indicate that there is a problem with deferring both *P* and *Q* until after time 10. However, such a decision would lead to failure: if the clock time reaches 11 and neither *P* nor *Q* has been executed, then all four solutions to the DTP will have been eliminated. Thus, in addition to the information in the ET, the dispatcher must also provide a second type of information to the executive. The ***deadline formula (DF)*** provides the executive with information about the next deadline that must be met.

In the next section, we explain how to calculate the DF, which is more complicated than computing the ET. Here we simply complete the example, by illustrating how the ET and the DF would be updated as time passes. The initial DF would indicate that either *P* or *Q* must be executed by time 10. Suppose that at time 8, action *P* is executed. At this point, STP_3 and STP_4 are no longer solutions. The ET then becomes { <*Q*, {[15,20]}>, <*R*, {[11,12], [21,22]}> } and the DF is trivially "*Q* by 20" . In this case, an update to ET and DF resulted because an activity occurred. However, updates may also be required when an activity does not occur within an allowable time window. For example, if *R* has still not executed at time 13, then its entry in the ET should be updated to be just the singleton [21,22], with no changes required to the DF. The example presented in this section contains variables with very little interaction. In general, there can be significantly more interaction amongst the temporal constraints, and the DF can be arbitrarily complex.

4 The Dispatch Algorithm

We now sketch our algorithm for the dispatch of plans encoded as DTPs. The input is a DTP and the output is an Execution Table (ET) and a Deadline Formula (DF). For each pair <*x*, TW(*x*)> in ET, *x* must be executed some time within TW(*x*). It is up to the executive to decide exactly when. The DF imposes the constraint that *F* has to hold by time *t*, where a variable that appears in the DF becomes true when its corresponding event is executed.

The dispatch algorithm will be called in three circumstances: (1) when a new plan needs to have its dispatch information initialized, at or before time *TR*; (2) when an event in the DTP is executed; (3) when an opportunity for execution passes because the clock time passes the upper bound of a convex interval in the time window for an action that has not yet been executed. Pseudo-code is provided in Figure 1. Space constraints preclude detailed description of the algorithm (but see [5]). Here we simply illustrate the procedure for computing the DF, the most interesting part of the algorithm.

Recall the example above. Initially, at time TR, the DTP has four solutions. To determine the initial DF, we consider the next critical moment, NC, which is the next time at which any action must be performed. This time is equal to the minimal value of all the upper bounds on time windows for actions, i.e., it is min{U(x,i)| x is an action in the DTP, and i is a solution STP}. For instance, in our example DTP, U(P, 1) = U(P, 2) = 10. The actions that may need to be executed by NC are those x such that U(x,i) = NC for some STP i. We create a list UMIN containing ordered pairs <x,i> such that U(x,i) = NC. In our current example, UMIN = {<P, 1>, <P, 2>, <Q, 3>, <Q, 4>}. Now we perform the interesting part of the computation. If <x,i> is in UMIN , it means that unless x is executed by time NC, STPi will cease to be a solution for the DTP. It is acceptable for STPi to be eliminated from the solution set only if there is at least one alternative STP that is not simultaneously eliminated. This is exactly what the deadline formula ensures: that at the next critical moment, the entire set of solutions will not be simultaneously eliminated. We thus use a minimal set cover algorithm to compute all sets of pairs <x,i> in UMIN such that the i values form a minimal cover of the set of solution STPs. In our example, there is only one minimal cover, namely the entire set UMIN. Thus, the initial DF specifies that P or Q must be executed by time 10: <P$\vee$ Q, 10>. In general, there may be multiple minimal covers of the solution STPs: in that case, each cover specifies a disjunction of actions that must be performed by the next critical time. For instance, suppose that some DTP has four solution STPs, and that at time TR, U (L, 1) = U (L, 2) = U (M, 3) = U (M, 4) = U (N, 4) = U (S, 3) = 10. Then by time 10 either L or M must be executed; additionally, at least one of L or N or S must be executed. The corresponding DF is <(L$\vee$ M)$\wedge$(L$\vee$ N$\vee$ S), 10>.

5 Formal Properties of the Algorithm

The role of a dispatcher is to notify the executive of when actions may be executed and when they must be executed. Informally, we will say that a dispatch algorithm is ***correct*** if, whenever the executive executes actions according to the dispatch notifications, the performance of those actions respects the temporal constraints of the underlying plan. Obviously, dispatch algorithms should be correct, but correctness is not enough. Dispatchers should also be ***deadlock-free*:** they should provide enough information so that the executive does not violate a constraint through inaction. A

```
Initial-Dispatch (DTP D)
  1. Find all n solutions (consistent component STPs) to D, calculate their distance
     graphs, and store them in Solutions [i]. Associate each solution with its (integer-
     valued) index.
  2. Set the variable TR to have the status Executed, and assign TR=0.
  3. Compute-Dispatch-Info(Solutions).

Update-for-Executed-Event (STP [i] Solutions)
  1. Let $x$ be the event that was just executed, at time $t$.
  2. Remove from Solutions all STPs $i$ for which $t \notin$ TW $(x,i)$.
  3. Propagate the constraint $t \leq x - TR \leq t$ in all remaining Solutions.
  4. Mark $x$ as Executed.
  5. Compute-Dispatch-Info (Solutions).

Update-for-Violated-Bounds (STP[i] Solutions)
  1. Let $U = \{U(x, k) | U(x, k) < Current\text{-}Time\}$
  2. Remove from Solutions all STPs $k$ that appear in $U$.
  3. Compute-Dispatch-Info (Solutions).

Compute-Dispatch-Info (STP[i] Solutions)
  1.     For each event $x$ in Solutions
  2.         {If $x$ is enabled
  3.         ET = ET $\cup$ $<x$, TW$(x)>$}.
  4.     Let U = the set of upper bounds on time windows, U$(x,i)$ for each still un-
         executed action x and each STP $i$.
  5.     Let $NC$, the next critical time point, be the value of the minimum upper
         bound in U.
  6.     Let $U_{MIN} = \{U(x, i) | U(x,i) = NC\}$.
  7.     For each $x$ such that $U(x,i) \in U_{MIN}$, let $S_x = \{i \mid U(x,i) \in U_{MIN}\}$
  8.         {Initialize F = true;
  9.         For each minimal solution MinCover of the set-cover problem (Solu-
             tions, $\cup S_x$), let $F = F \wedge (\vee_{x \mid S_x \in MinCover}\, x)$.
  10.    DF = $<F, NC>$.}
```

Figure 1. The Dispatch Algorithm

third desirable property for dispatchers is ***maximal flexibility:*** they should not issue a notification that unnecessarily eliminates a possible execution, i.e., an execution that respects the constraints of the underlying plan. Finally, we will require dispatch algorithms to be ***useful***, in the sense that they really do some work. Usefulness will be defined as producing outputs that require only polynomial-time reasoning on the part of the executive. Without a requirement of usefulness, one could achieve the other three properties by designing a DTP dispatcher that simply passed the DTP representation of a plan on to the executive, letting it do all the reasoning about when to execute actions.

Our dispatch algorithm has these four properties, as proved in [5]. The proofs depend on a more precise notion of how the dispatcher and the executive interact. The dispatcher issues a *notification sequence,* a list of pairs $<\text{ET,DF}>_1 \ldots ,<\text{ET,DF}>_n$, with a new notification issued every time an event is executed or an upper bound is passed. The executive performs an *execution sequence*, a list $x_1= t_1, \ldots, x_n=t_n$ indicating that event x_i is executed at time t_i, subject to the restriction that $j>i \Rightarrow t_j > t_i$. An execution sequence is complete if it includes an assignment for each event in the original DTP; otherwise it is partial. The notification and execution sequences will be interleaved in an *event sequence*. We associate each execution event with the preceding notification, writing Notif(x_i) to denote the notification of event x_i.

Definition. An execution sequence *E respects* a notification sequence *N iff*

1. For each execution event $x_i=t_i$ in *E*, $<x_i$, TW $(x_i)>$ appears in ET of Notif (x_i) and $t_i \in$ TW(x_i), i.e., each event is performed in its allowable time window.
2. For each DF=$<F,t>$ in *N*, $\{x_i \mid x_i = t_i \in E$ and $t_i \leq t\}$ satisfies *F*. That is, the execution sequence satisfies all the deadline formulae.

Theorem 1: The dispatch algorithm in Fig. 1 is correct, i.e., any complete execution sequence that respects its notifications also satisfies the constraints of D.
Theorem 2: The dispatch algorithm in Fig. 1 is deadlock-free, i.e., any partial execution that respects its notifications can be extended to a complete execution that satisfies the constraints of D.
Theorem 3: The dispatch algorithm in Fig. 1 is maximally flexible, i.e., every complete execution sequence that respects the constraints in *D* will be part of some complete event sequence.
Theorem 4: The dispatch algorithm in Fig. 1 is useful, i.e., generating an execution sequence is polynomial in the size of the notifications.

References

1. Muscettola, N., P. Morris, and I. Tsamardinos. Reformulating Temporal Plans for Efficient Execution. in Proceedings of the 6th Conference on Principles of Knowledge Representation and Reasoning. 1998.
2. Tsamardinos, I., P. Morris, and N. Muscettola, Fast Transformation of Temporal Plans for Efficient Execution, in Proceedings of the 15th National Conference on Artificial Intelligence. 1988, AAAI Press/MIT Press: Menlo Park, CA. p. 254-261.
3. Wallace, R.J. and E.C. Freuder, Dispatchable Execution of Schedules Involving Consumable Resources, in Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling. 2000.
4. Dechter, R., I. Meiri, and J. Pearl, Temporal Constraint Networks. Artificial Intelligence, 1991. 49: p. 61-95.
5. Tsamardinos, I., Constraint-Based Temporal Reasoning Algorithms, with Applications to Planning. 2001.

APPENDIX F

A Scheme for Integrating E-Services in Establishing Virtual Enterprises*

Alan Berfield, Panos K. Chrysanthis
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260, USA
{alandale,panos}@cs.pitt.edu

Ioannis Tsamardinos
Intelligent Systems Program
University of Pittsburgh
Pittsburgh, PA 15260, USA
tsamard@cs.pitt.edu

Martha E. Pollack
Department of EE and Computer Science
University of Michigan
Ann Arbor, MI 48109, USA
pollackm@eecs.umich.edu

Sujata Banerjee‡
Info. Sci. & Telecom. Dept.
University of Pittsburgh
Pittsburgh, PA 15260, USA
sujata@tele.pitt.edu

Abstract

An important aspect of Business to Business E-Commerce is the agile Virtual Enterprise (VE). VEs are established when existing enterprises dynamically form temporary alliances, joining their business in order to share their costs, skills and resources in supporting certain activities. Currently, existing enterprises use workflows to automate their operation and integrate their information systems and human resources. Thus, the establishment of a VE has been viewed as a problem of dynamically expanding and integrating workflows. In this paper, we present an approach to combining workflows from different enterprises, using techniques developed in the Artificial Intelligence literature on planning. Our method takes two workflow views, one representing a service request and the other a service provision (advertisement), with a mix of vital and nonvital steps and a rich set of constraints, and returns a list of possible legal combinations, if any exist. It then uses plan-merging techniques to find potential conflicts between the two workflows, and to suggest additional constraints that can resolve the conflicts. The returned solutions represent terms for the establishment of a new VE, and can be evaluated by each side to determine which is most desirable.

*This material is based upon work partially supported by NSF IIS-9812532 and AFOSR F30602-00-0547 awards.

‡On Leave of Absence at Hewlett-Packard Laboratories, MS 1U-17, Palo Alto, CA 94304, USA.

1. Introduction and Motivation

Electronic Commerce is expanding from the simple notion of E-Store to the notion of *Virtual Enterprises* (VEs) where existing enterprises dynamically form temporary alliances, joining their business in order to share their costs, skills and resources in supporting certain activities. An example of a VE in the context of the travel industry would be the collaboration of different travel agents, airliners, ground transportation services, hotels, restaurants and entertainment services in order to set up and manage a tourism trip.

Many enterprises use workflows to automate their operation and integrate their information systems and human resources [19]. A workflow consists of a set of *activities* (also called *tasks*) that need to be executed according to given temporal constraints over a combination of heterogeneous database systems and legacy systems. A major challenge has been the development of *workflow management* systems (e.g., [9, 5, 13, 1]). Several techniques have been developed for correct and reliable specification, execution, and monitoring of workflows and the involved external support. Many of these techniques are extensions of those in transaction processing in databases combined with general middleware services such as those found in CORBA/DCOM and more recently in Java-based services such as Jini from Sun and E*Speak from HP.

Very recently the idea of the use of workflows to support multi-organizational processes that form a virtual enterprise has attracted some attention [10, 6, 8]. The establishment of a VE can be seen as a problem of dynamically expanding and integrating workflows in decentralized, autonomous and interacting workflow management systems

[2, 7, 12]. During the establishment of a VE, a distributed, multi-organizational workflow emerges from the dynamic merging and reconfiguration of workflows representing E-Services in the participating enterprises. In our previous work, we looked at using mobile agents as a platform for advertising, negotiating, and exchanging control information about E-Services for the establishment of VE's [6]. In this paper, we focus on a method for verifying that the VE is compatible with workflows in the participating enterprises.

The contribution of this paper is a new method for establishing VEs, involving both the generation of outsourcing requests and the validation of constraints. The scheme incorporates techniques developed in the Artificial Intelligence (AI) literature on planning, specifically algorithms for merging temporal plans. Within the AI literature, a plan is a collection of steps (i.e., tasks), with causal, temporal, and resource constraints. A plan is intended to represent a course of action that will achieve a specified goal when executed beginning in a specified initial state. Critical to the notion of plans is that of *causal structure*: the steps in each plan are specified in terms of their preconditions and effects, and the plan records information about which steps cause (or *establish*) the preconditions of other steps. When merging together two plans, it is necessary not only to check that there are no violations of the temporal and resource constraints of the plans being merged, but also to ensure that the necessary causal relations are maintained in the merged plan. We argue in this paper that similar consistency requirements also hold when a VE is formed.

In the next section, we review the basic structures used in workflows. In particular, we describe a class of workflows that include specifications of preconditions and effects. In Section 3, we describe a VE and its components. Section 4 describes our detailed scheme for establishing such a VE. Section 5 deals with the current state of our implementation. We conclude with a summary in Section 6.

2. Workflow Model

Workflows encode tasks and the relationships among them. Workflow specification formalisms generally provide a small set of basic control flow relationships among tasks. Typically there are four such relationships: OR-split, AND-split, OR-join and AND-join. The first two relationships are used to specify branching decisions in a workflow whereas the remaining two specify points where activities converge to initiate the next activity within a workflow. An OR-join specifies alternatives whereas AND-join specifies required activities.

While the relations just listed provide information about the relative ordering of the tasks in a given workflow, to handle the problem of forming Virtual Enterprises, it is also necessary for the workflow to model a significant amount of information about each task. Thus, we will assume an enriched model of workflows in which each task has the following information associated with it:

- *Pre- and postconditions*, which specify what must be true before a task can be executed, and what will be made true as a result of the task's performance.
- *Causal links*, which relate each task that establishes a condition (listed in its postconditions) to the task that requires it (listed in its preconditions).
- *In- and out-parameters*, which are used in the evaluation of preconditions and postconditions. They carry information and engender data flow during execution. For example, a credit card number could be an in-parameter to a "pay for dinner" task.
- *Temporal Constraints* that specify the earliest and latest start and end times of a task, as well as the minimal and maximal durations of the task. They can be absolute times or relative to the execution of other tasks.
- *Resource Constraints*, which specify the equipment, material, or agent resources required for the task.
- *Significance*, which indicates whether the task is *vital* to the workflow and therefore must be executed, or whether it is *nonvital*, and need only be executed if feasible [6].
- *Cost*, which represents the price of the task.

Other information may also be associated with each task, such as rules for exception handling should the task fail. However, we will not be concerned with these types of information in the current paper.

As shown in Figure 1, a workflow can be graphically depicted with nodes (thick boxes) denoting activities and arrows denoting precedence. The figure represents a business trip from [15]. Shaded nodes indicate vital activities that must be completed to ensure proper execution. Nodes with a pair of dashed lines leading to another workflow are hierarchical activities: those that can be decomposed into workflows themselves. AND-splits and AND-joins are represented implicitly when two or more causal links emanate or arrive at a node respectively. To represent OR-splits we insert a conditional node that creates two new execution contexts (branches), e.g., one for success and one for failure (in Figure 1, these are shown as nodes with edges labeled S and F). Tasks are executed only when their context is true. The OR-join is represented implicitly when the context S or F disappears from the labels of subsequent edges.

We are assuming a typical Workflow Management System (WfMS) architecture with our enriched model of workflows. Specifically, a WfMS consists of the following three basic components:

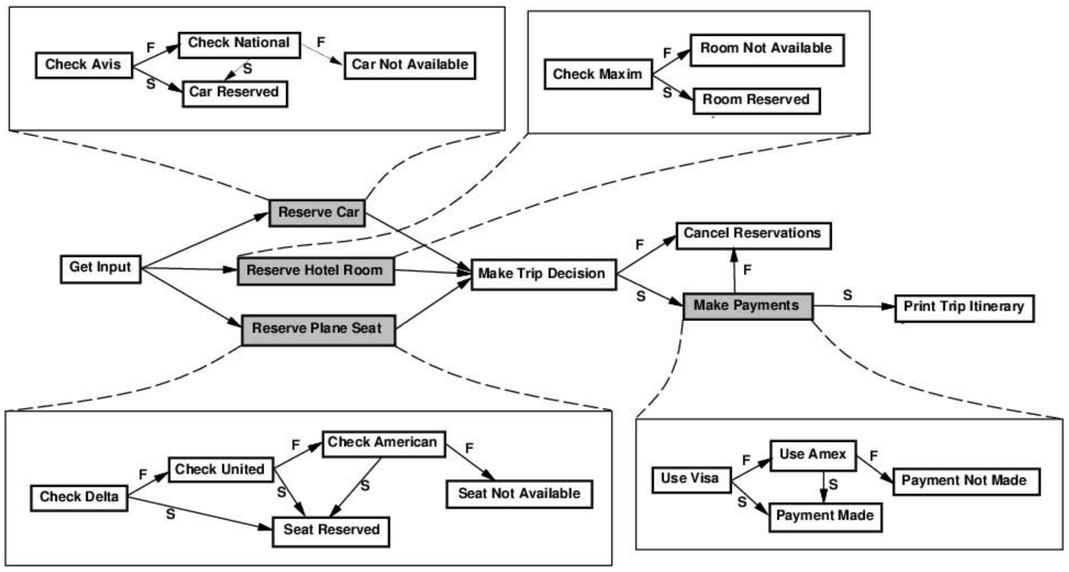


Figure 1. Trip Plan Workflow

- *Workflow Schema Library*, which contains workflow schemas or templates and generic constraints.
- *WfMS Services*, functions provided by the WfMS for managing workflows. These include specifying workflows, verifying their correctness, instantiating and scheduling them, executing them, and monitoring their execution.
- *Workflow Repository*, which contains all instantiated and scheduled workflows, i.e., the workflows the business is committed to performing.

3. Forming Virtual Enterprises

A Virtual Enterprise (VE) is formed when a business decides to commit to a new workflow, while outsourcing some of the work involved in that workflow. Consider the example of Jane Smith, an executive planning a trip to Vienna. She gets in touch with a travel agency to arrange the trip. She decides that while she is there she would like to attend an opera and tour the Art Museum. This adds the nonvital nodes “buy opera ticket” and “buy museum tour ticket” to the trip schema (Figure 1). The travel agency lacks connections with the entertainment/opera industry, so is unable to purchase such tickets. In order to satisfy the customer, they decide to outsource those tasks.

The above example represents a common reason for outsourcing. When a business receives a new request from a client, it takes the form of an instantiated workflow schema from its Workflow Schema Library. The client may have added constraints and/or customized the schema by adding new nodes, which could represent extra or special activities and opportunities. The business may select some of the tasks from the workflow to outsource and/or it may select some of the open conditions from the workflow and outsource their achievement. This outsourcing establishes a VE.

In our VE workflow specification, we use the notion of views to express outsourcing. Any subgraph of a workflow graph defines a *segment* or a *view* of the workflow. Formally, a workflow view can be defined as a *projection* on the graph based on some criteria ($projection(workflow, <criteria>)$). For example, consider the view that includes all and only the vital nodes of the full workflow. The requirement that nodes be vital is the criteria used by the projection.

$$VitalView = projection(workflow, \{a \mid a \in workflow \wedge a.significance = vital\})$$

The nodes in a view retain all information of their originals, including all constraints. However, because all constraints are maintained, a view may have nodes that have temporal constraints referring to other nodes not actually in the view, and may also have broken causal links possibly resulting in unsatisfied preconditions (i.e., a node in the view could have a precondition that was established by some node in the full workflow that is not in the view).

A workflow view can represent any activity performed by a *service provider* on behalf of a *service requester*. Consequently, workflow views can be used to express service requests or service provision (advertisement). In our proposed system, it is these workflow views that are being requested and advertised.

In our scheme, a request has the following structure:

Requests: $Rq = (P, G, RW)$
where P = Service Requester Profile,
G = set of Goals,
RW = Requested Workflow View

The profile can contain various information about the requester, such as name, site identification, credentials, etc.. It may also contain a target price range. The set of goals is a list of all goals (postconditions) that need to be accomplished. The workflow view captures all temporal constraints and resource usage issues involved.

In the example above, the request includes the profile of the travel agency, the goals "opera ticket purchased" and "museum ticket purchased", and a view with two nodes that indicate times by which the tickets must be purchased.

A requested workflow view can be potentially augmented during negotiation to match the service provider's workflow, reflecting opportunities, omitted activities and data. During a negotiation we may decompose the required view into several views and seek other service providers for the other parts of the view. In this way, a single initial request may lead to the establishment of a VE comprising multiple enterprises. A VE comprising multiple enterprises can also result when a service provider's view includes outsourcing. We will elaborate on this in the next section.

The structure of an advertisement is the same as that of a request.

Advertisements: $Ad = (P, G, AW)$
where P = Service Provider Profile,
G = set of Goals,
AW = Advertised Workflow View

It includes a profile, the set of goals accomplished, and a workflow view encompassing constraints. The profile, in addition to other information, may contain cost information for the workflow as a whole, such as minimum cost, maximum cost (cost with all nonvital steps), or both. The list of goals indicates what the advertised workflow actually does, and may also include goals associated with nonvital activities. Such advertisements will typically be stored in the databases of trading servers. Each provider may have a set of advertisements with the same goals but with a different associated workflow view (i.e., different constraints).

To return to our example, an advertisement that would be of interest to the travel agency would be for a business that specializes in Vienna cultural events, including opera. The single goal "opera ticket purchased" is accomplished. Its workflow view includes the tasks "contact opera houses", "read current reviews", and "purchase ticket."

The VE environment is a distributed environment. It consists of multiple businesses, acting as requesters and providers, using services provided by negotiation areas or trading places. The trading places could contain databases of advertisements and could provide services allowing businesses to both place advertisements and to find advertisements that meet their goals. Standardization of representation is clearly required (particularly of preconditions, effects, and goals), and could be enforced by the trading servers. A portion of this environment is shown in Figure 2. Two negotiation areas are depicted, as well as four businesses' WfMS. Shaded nodes again represent vital activities, and an advertised hierarchical Opera activity is shown partially expanded.

3.1. Commitment and Outsourcing Request Generation

In order to commit a new, possibly customized workflow, a WfMS needs to make sure that it is schedulable. A workflow is schedulable if it is *correct*, *complete*, and *compatible* with existing commitments.

Definition 1 Workflow Correctness: *A workflow is* correct *if and only if*

1. *it has no conflicting temporal or resource constraints,*
2. *for each goal/precondition P, there is a task that achieves P (the producer task), and it is ordered before the task that requires it (the consumer task), and*
3. *for each goal/precondition P, no task that may negate P can possibly be ordered in between the producer and the consumer.*

This notion of correctness is important as only correct workflows can possibly be executed. Note that some workflows may contain preconditions that are assumed to be established independently of the workflow itself. We will call such preconditions *open* with respect to the workflow. A simple example of such an open precondition is a workflow for renting a car that assumes the precondition of having a driver's license. Workflows with such open preconditions are incorrect until they have been combined with other workflows that establish all open preconditions.

Definition 2 Workflow Completeness: *A* complete *workflow is a workflow that specifies all tasks needed to achieve its goals and preconditions.*

Definition 3 Workflow Compatibility: *A workflow is* compatible *with another if none of its nodes conflict with any of the other's (and vice versa).*

This means that the temporal constraints, resource usage, and postconditions of its nodes do not prevent the execution of the nodes in the other workflow (though they may place limits on when those nodes can be executed). So for example, a compatibility conflict between workflows arises if

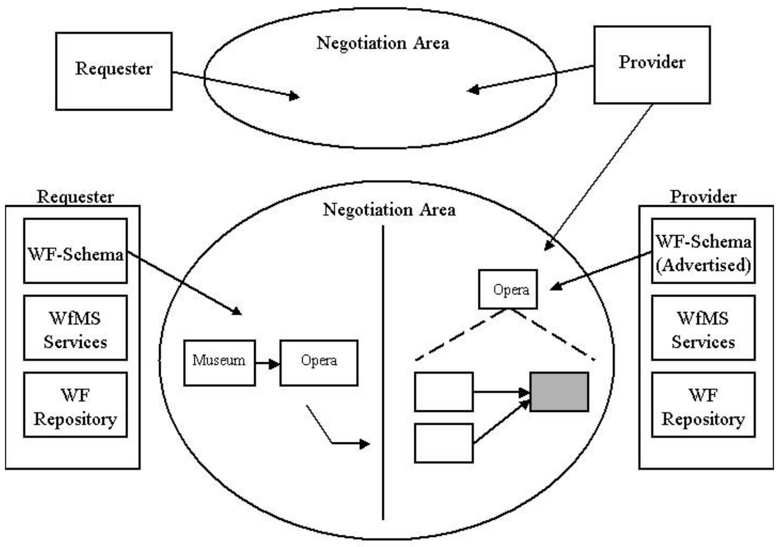


Figure 2. VE Environment

two tasks that use the same resource (e.g., equipment) are set to execute at the same time. Another example is a task that dictates that a robot move to the printing room for the purpose of getting a faxed itinerary, which conflicts with a task that moves the robot to another room that could be executed after going to the printing room but before fetching the fax.

An alternative definition of the compatibility of two workflows is that the workflow resulting from their union is correct. We propose the notion of a *merge* with the Workflow Repository for determining the compatibility of a workflow with the currently scheduled workflows (in the Repository). If the merge is successful, the new workflow can be committed and its execution enabled. If the merge is unsuccessful, the new workflow is not compatible and the business may consider outsourcing.

An effective merging process will check whether the above requirements (correct, complete, and compatible) are met, and will indicate where problems lie: what nodes are conflicting with others, which have unsatisfied (open) preconditions, or which the business lacks the necessary expertise (i.e., roles as resources) to accomplish. It may also suggest additional temporal or resource constraints that are required to ensure that they are met. However, it's desirable to impose a minimal set of extra constraints, i.e., to provide a least-commitment response, as this allows increased flexibility to respond to changes that may arise during execution.

The merge process can also be used to identify and construct outsourcing requests. In the event of an unsuccessful merge, any nodes from the new view that are indicated as problems by the merging process (those having irresolvable resource conflicts with existing commitments) will form part of the requested workflow view *VRq* by extracting them from the full workflow using projection. In addition to these nodes, for each open precondition in the new view not satisfied by the existing commitments (such preconditions will be found by the merge process), a new place-holder node is added to the view. Each of these new nodes represents a task that accomplishes one of the open preconditions, i.e., it has one of the open preconditions set as its postcondition, and any associated temporal and causal links are applied. The complete set of postconditions of every node in *VRq* make up the goal set of the request, G. In the simplest case *VRq* would be a single node, with associated constraints. More complex cases would involve multiple nodes and richer constraints.

Recall that projected nodes maintain all constraints and conditions they had in the parent workflow, and may therefore include unsatisfied preconditions and temporal constraints referring to nodes not in the view. This is not really a problem as they will be satisfied by non-outsourced nodes. The preconditions, along with in-parameters, represent the input to the outsourced view. Goals and out-parameters of the outsourced nodes represent the output.

4. Outsourcing Scheme

In this section, we discuss in detail the steps for outsourcing and establishing a VE.

Let R be a Requester and P be a set of providers $\{P_1, ..., P_n\}$. R has a set of workflows to which it is already committed, and which it stores in the WF Repository;

let us call them *CR* (commitment workflows at requester). Similarly, each P_i has a set of workflows already committed to; let us call them $CP(i)$.

Let $Rq = (R, G, VRq)$ be a request of R for outsourcing with goals $G = \{G_i, ...G_n\}$ and workflow view *VRq*. Each P_i can provide a set of alternative workflow views $A(P_i)$ for achieving one or more G_j of Rq.

The problem of outsourcing is how to pick a set of workflow views S from the $A(P_i)$ of one or more P_i so that the combined set satisfies Rq and merges with *CR*, and each $A(P_i)$ in it merges with its provider's $CP(i)$. Specifically, such a set achieves all goals of the outsourced workflow, all temporal constraints are satisfiable, there are no resource conflicts, and for every precondition of every workflow activity in *CR* and $CP(i)$ there exists a causal dependency that ensures that the precondition will be met.

Formally, we want a set $S = wf_1 \cup wf_2 \cup wf_3 \cup ... \cup wf_n$, where $wf_j \in \bigcup_i A(P_i),\ j = 1, \ldots, n$ such that

- $postconditions(S) \supseteq G$
  (all outsourcing goals are met)
- $\forall wf_x \in S\ \ postconditions(wf_x) \cap G \neq \phi$
  (each workflow achieves at least one goal)
- $compatible(S, CR)$
  (S is compatible with the requester's commitments)
- $\forall wf_x \in A(Py)\ compatible(wf_x, CP(y))$
  (each alternative workflow is compatible with its provider's commitments)

The above suggests a solution that has three phases:

1. Finding a set of alternative workflows that satisfy Rq
   (*Terms for the Establishment of a VE*)
2. Checking for the satisfaction of $CP(i)$
   (*Providers Validation of Terms and E-Service Bids*)
3. Check for the satisfaction of R
   (*E-Service Bid Evaluation*)

We elaborate on these phases in the next subsections.

4.1. Phase 1: Terms for the Establishment of a VE

As mentioned previously, we assume in this paper that finding alternative workflow views that satisfy a request *Rq* is a service provided by trading servers. Each alternative view represents a term for the establishment of a VE. During this first phase the sets $A(P_i)$ of alternative workflow views are generated. These views accomplish the goals G of *Rq* while not violating any of its constraints. For the sake of simplicity, we will assume in the rest of our discussion that there is only one trading server.

The service searches the database of the trading server, looking for advertisements that meet some or all of the requested goals. Which advertisements are examined first depends on the selection conditions being used. One such condition would include the desirability of first considering those that accomplish all goals, and only considering multiple, partial matches when all such are found. For each advertisement found and selected, the server must finally determine if it or any of its alternatives (involving different combinations of nonvital nodes) can meet the constraints of the request. This process continues until all advertisements that meet any goals have been examined, or some termination criteria are met (such as a deadline for search time).

For the detailed explanation, we will consider one such advertisement found and selected by the trading server that accomplishes all goals in G; let us call it $Ad1$.

As shown in Figure 3, the service must determine if $Ad1$ will satisfy the constraints in the request's workflow view Rq. To do this, $Ad1$ and Rq are first stripped down to only vital nodes using projection. Temporal constraints of the vital nodes may need to be adjusted, as any referring to nonvital nodes will be invalid. For any node that has such a constraint, there are four possible situations:

1. The nonvital node referred to has no constraints on its time[1] : the constraint on the vital node can be dropped.
2. The nonvital node has an absolute time: that time can be used.
3. The nonvital node has a time relative to some other vital node: the reference to that node can be used.
4. The nonvital node has a time relative to some other nonvital node: that nonvital must be searched in the same fashion for a time or vital node reference.

It may be beneficial to instead store such alternative constraints with the vital nodes in order to save computational time, though the number of nonvital nodes is likely to be small.

Next the service attempts to bind the constraints of the vital-only view of Rq (called RqV in the figure) to the stripped view of $Ad1$ (AdV in the figure). Binding adds the constraints of the requested nodes to the corresponding nodes in the advertisement (those that have the same postconditions). If AdV cannot support the added constraints (because they conflict with existing ones), the bind fails and the function must backtrack to find a different advertisement. Otherwise, the new bound advertised view BV is added to $A(P_i)$, where P_i is the provider of BV, and the search continues for its variants that include nonvital nodes.

The search for variants of BV considers combinations of BV and nonvital nodes from the full $Ad1$ and Rq. This can be achieved by the function *addNodes*, that adds a group of nodes to the workflow *BV*, restoring any modified temporal constraints that referred to them. This is basically a merge process. The addNodes function fails and returns null if the

[1] By "time" we mean either start or end time of the task, depending on which the specific temporal constraint refers to.

resulting view is incorrect (i.e., the new nodes cannot be added without violating constraints). If the function does not fail, the resulting view is added to $A(P_i)$.

It is interesting to point out that there is another possible method for this search: working in the other direction, starting by adding back all nonvitals and then removing them to find correct alternatives. It is not clear which approach is better, but we intend to investigate this in future work.

In either case, the search will proceed until all possible combinations have been attempted or some other termination criteria have been reached. As most views are expected to have 3 or fewer nonvital steps, finding all possible combinations is not likely to be impractical. Once the search has finished, $A(P_i)$ contains every alternative workflow solution for the workflow $Ad1$.

The service generates a set $A(P_i)$ for each P_i with at least one selected advertisement. The $A(P_i)$'s created in this fashion are now sent out to their respective provider for validation.

4.2. Phase 2: Providers Validation of Terms and E-Service Bids

The second phase of the outsourcing takes place at the providers of the advertisements. Each P_i receives the $A(P_i)$ generated for it in the previous phase, and must determine whether any of the workflow views in $A(P_i)$ are compatible with its $CP(i)$. Each alternative basically represents a potential new incoming workflow to be scheduled. Recall that such scheduling can be accomplished using the merge process. Thus, the provider attempts to merge each alternative with $CP(i)$ independently. Any that fail are removed from $A(P_i)$. Those that succeed can be kept to form the basis for the service bid. Of course, if $A(P_i)$ is empty at the end of this phase, then none of the views were compatible with the provider's commitments.

To generate the full service bid, each view remaining in $A(P_i)$ could possibly be expanded into multiple views if the provider wishes to add additional nonvital nodes (representing special offers or bonuses). Note that such additions would likely increase cost, but would possibly also increase value. The provider may also rank the solutions in order of preference or cost to help the later decision process.

Each provider sends its service bid to the requester to be evaluated in the next phase.

4.3. Phase 3: E-Service Bid Evaluation

In the third phase, the requester evaluates and selects an E-Service bid. Of the views in the service bids returned by the providers, the requester must determine which are compatible with its *CR*. This is done in exactly the same fashion as with the providers.

Each service bid in the returned list is combined with the rest of the original workflow to form a complete solution. For each solution, a merge is attempted with the committed workflows. Any that fail to merge are discarded. Those that successfully merge are correct views that each accomplish the outsourcing and that are compatible with both the provider's and requester's previous commitments.

The requester may then evaluate these remaining views to make a final decision as to which one will be used, which likely involves cost comparisons.

4.4. Multiple Partial-Solution Views

In the previous discussion, we assumed the simplest case where there exist advertised views that accomplish all the goals of the request. However, in many cases there may be no single advertisements that accomplish them all. This would require views from multiple advertisements to be combined in order to meet the requester's needs. In order to handle these cases, the described first and third phases need to be enhanced.

For example, in Phase 1, the search for alternatives must also search for combinations of advertisements that accomplish all goals. The merge process can be used again to verify that these combinations of advertisements are compatible with each other in addition to meeting the constraints of the request. For combined views belonging to a single provider, the combination (and its alternatives involving nonvital nodes) are grouped together as a single view.

The requester in Phase 3 must be aware that returned views do not necessarily accomplish all goals. Any E-Service bids that only satisfy some of the goals must be combined with other returned views to form complete solutions.

5. Implementation

In our previous work, we proposed to use mobile agents as a platform for establishing VE's [6]. Our goal is to implement our scheme described in the previous section on this platform. The idea is to use mobile agents to perform the phases of the scheme. The requester dispatches an agent with its request. The agent visits trading servers, and spawns copies of itself to deliver alternatives to different providers. It then gathers all returned service bids together and delivers the results back to the requester.

A core concept in our scheme for integrating E-Services is the merge process. It is this process that verifies whether or not different workflow views are compatible with each other. It is also responsible for adding nonvital nodes to views and verifying that a view is compatible with a business' existing workflow repository. The merge process can even be used to generate the outsourcing requests.

```
Repeat
    • $Ad1 = Ad \in DB \mid \; < selection\ conditions >$
    • $AdV = projection(Ad1, \{a \mid a \in Ad1 \wedge a.significance = vital\})$
    • $RqV = projection(Rq, \{a \mid a \in Rq \wedge a.significance = vital\})$
    • $BV = bind(AdV, RqV)$
    • Repeat
        – $Nodes = projection(Ad1, selected\ nonvital \in Ad1) \; \cup \; projection(Rq, selected\ nonvital \in Rq)$
        – $Bx = addNodes(BV, Nodes)$
        – $IfBx \neq null \rightarrow A(P_i) = A(P_i) \cup Bx$
    • Until all combinations of nonvitals found or termination criteria met
Until all Ads found that meet $< selection\ conditions >$ or termination criteria met
```

Figure 3. Service For VE Terms

Merging is not a trivial problem. It can be formulated as a *Constraint Satisfaction Problem* or CSP, with temporal features. The process must consider temporal constraints, resource usage, and causal links (preconditions and effects). There has been a great deal of research done on similar problems by the Artificial Intelligence community [14, 17, 20]. A number of formalizations have been developed for variations with more or less expressivity. The two that most closely match our problem are the Disjunctive Temporal Problem (DTP) and the Conditional Disjunctive Temporal Problem (CDTP).

For solving DTPs we have developed and implemented a new algorithm called Epilitis [16], along with algorithms that convert CDTPs to DTPs so that they may be solved by it as well. Epilitis builds on plan merging techniques used in a tool called PMA (Plan Management Agent) [18]. Epilitis integrates a number of techniques for pruning the search space, some of which are Conflict Directed Backjumping, Removal of Subsumed Variables, Semantic Branching, and no-good learning. Epilitis is currently the most efficient algorithm for solving such problems, as experimental results have shown that it is two orders of magnitude faster than the previous state-of-the-art solver, on synthetic benchmark problems.

In the prototype system we are currently developing, we will use Epilitis for the merging process at the WfMS and the trading servers. The representation that Epilitis expects is nearly identical to our enhanced model of workflows; the mapping between the two is trivial. Merging with Epilitis has all the properties discussed in Section 3. Any conflicting tasks are identified with explanation, and a minimal number of constraints are added. Plans with disjunctive temporal constraints are supported (for added flexibility), and duplicate nodes can be identified and pruned/combined.

Epilitis does not support the notion of significance (vital vs. nonvital tasks). However these are implemented in the higher-level layer that performs the phases of our scheme. Only this layer is aware of the vital/nonvital distinction. (This is the cause of some of the complexity in the search for alternatives, as all the different combinations of nonvitals must be attempted separately.) This layer also serves to interface Epilitis with a relational DBMS using Microsoft Access and MySQL that will be used to implement the Workflow Repositories.

The current version of Epilitis is written in LISP, but using JLinker we have interfaced it to the rest of our prototype which is being developed in Java. The new version of Epilitis currently being developed will be in Java as well.

6. Conclusions

We are concerned with integrating E-Services for the establishment of a VE, where such services are represented with workflows. We have therefore created algorithms that make use of existing plan merging and temporal reasoning algorithms from the AI literature. Our scheme is sound, in that the workflows it returns as possible merge candidates are guaranteed to be correct. It is also complete, in that it will find all such candidates, given sufficient time. It further ensures that the merge candidates are compatible with all businesses involved in the VE. It can create the outsourcing requests based on identified conflicts, handle any number of nodes and workflows to be outsourced, and is flexible in that it can build a VE using multiple providers, each with their own set of constraints. Our scheme also takes into consideration that workflows have both vital and nonvital steps, and appropriately considers them in its search.

In our proposed system, the merging process is built with existing AI algorithms. The specific algorithm, Epilitis, is the best algorithm available at this time. It has been implemented and is a fully functional and working plan merging tool. Currently we are developing our prototype system. Our goal is to evaluate its performance in terms of speed and memory usage. Another area we intend to explore is its use as a plan/workflow repair system that would replace broken or invalidated nodes or views with alternatives, possibly located in different databases on various machines.

Recently, there have been a variety of platforms developed with business to business E-services and Virtual Enterprises in mind. E*speak [3] from HP, VorteXML [4], and CrossFlow [11] are examples. These systems provide various features for managing and monitoring VE's, along with some standards for communication. Such systems could potentially be augmented or used conjunctively with our scheme for automated VE establishment. We will investigate such possibilities as part of our future work.

References

[1] Alonso G., D. Agrawal, A. El Abbadi and C. Mohan. Functionalities and Limitations of Current Workflow Systems. *IEEE Expert*, 12(5), 1997.

[2] Casati F., S. Ceri, B. Pernici and G. Pozzi. Workflow Evolution, *Data & Knowledge Engineering*, 24(3): 211-238, 1998.

[3] Casati F. and M. Shan. Definition, Execution, Analysis, and Optimization of Composite E-Services. Bulletin of the Technical Committee on Data Engineering, 24(1):30–35, 2001.

[4] Christophides V., R. Hull, A. Kumar, and J. Simeon. Workflow Mediation using VorteXML. Bulletin of the Technical Committee on Data Engineering, 24(1):41–46, 2001.

[5] Chrysanthis P. K.. Guest Editor's Introduction to Special Issue on Workflow Systems. *Distributed Systems Engineering*, 3(4):211–212, 1996.

[6] Chrysanthis P., T. Znati, S. Banerjee, S. Chang. Establishing Virtual Enterprises by means of Mobile Agents. *Research Issues in Data Engineering*, 1999.

[7] Cichocki A. and M. Rusinkiewicz. Migrating Workflows. *Workflow Management Systems and Interoperability*, A.Dogac et al. (Eds)Springer Verlag, Series F, Vol 164, pp. 339–355, 1998.

[8] Davulcu H., M. Kifer, L. R. Pokorny, C. R. Ramakrishnan, I. V. Ramakrishnan, and S. Dawson. Modeling and Analysis of Interactions in Virtual Enterprises. *Research Issues in Data Engineering*, 1999.

[9] Georgakopoulos D., M. Hornick and A. Sheth. "An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure." *Distributed and Parallel Databases*, 3(2), 1995.

[10] Georgakopoulos D., H. Sinha, K. Huff and B. Hurwitz. "Monitoring Multi-organizational Processes." *Proc. of the 11th Int'l Conf. on Parallel and Distributed Computing Systems*, pp. 75–80, 1998.

[11] Grefen P., K. Aberer, H. Ludwig, and Y. Hoffner. CrossFlow: Cross-Organizational Workflow Management for Service Outsourcing in Dynamic Virtual Enterprises. Bulletin of the Technical Committee on Data Engineering, 24(1):53–58, 2001.

[12] Han Y. and A. Sheth. On Adaptive Workflow Modeling. *Proc. of the 4th Int'l Conf. on Information Systems Analysis and Synthesis*, pp. 108-116, 1998

[13] Jablonski S. et al. External and Internal Support Services in Workflow Management Systems. *Proc. of the 11th Int'l Conf. on Parallel and Distributed Computing Systems*, pp. 81–86, 1998.

[14] V. Kumar. Algorithms for Constraint-Satisfaction Problems: A Survey. *AI Magazine*, 13(1):32–44, 1992.

[15] Ramamritham K. and P. K. Chrysanthis. *Advances in Concurrency Control and Transaction Processing*, IEEE Computer Society Press, 1997.

[16] Tsamardinos I. Constraint-Based Temporal Reasoning Algorithms with Applications to Planning. *Ph.D. Thesis. University of Pittsburgh Intelligent Systems Program*, 2001

[17] Tsamardinos I., M. E. Pollack, et al. Merging Plans with Quantitative Temporal Constraints, Temporally Extended Actions, and Conditional Branches. *Artificial Intelligence Planning and Scheduling (AIPS'00)*, Breckenridge, Colorado, USA, 2000

[18] Tsamardinos I., M.E. Pollack, et al. Adjustable Autonomy for a Plan Management Agent. *AAAI Spring Symposium on Adjustable Agents.*, 1999.

[19] Workflow Management Coalition, Technology & Glossary, Document Number WFMC-TC-1011, June 1996.

[20] Q. Yang. Intelligent Planning: A Decomposition and Abstraction Based Approach. Springer, 1997.