Merging Plans with Quantitative Temporal Constraints, Temporally Extended Actions, and Conditional Branches

Ioannis Tsamardinos
Intelligent Systems Program
University of Pittsburgh
tsamard@cs.pitt.edu

Martha E. Pollack
Dept. of Computer Science and
Intelligent Systems Program
University of Pittsburgh
pollack@cs.pitt.edu

John F. Horty
Philosophy Department and
Inst. for Adv. Computer Studies
University of Maryland
horty@umiacs.umd.edu

Abstract

We develop an algorithm for merging plans that are represented in a richly expressive language. Specifically, we are concerned with plans that have (i) quantitative temporal constraints, (ii) actions that are not instantaneous, but rather have temporal extent, and (iii) conditional branches. Given a set $\mathcal{S}$ of such plans, our algorithm finds a set of constraints that jointly ensure that the plans in $\mathcal{S}$ are mutually consistent, if such a set of constraints exists. The algorithm has three phases. In the first, it employs a new data structure, a *conditional simple temporal network* (CSTN), to identify conflicts between the plans. Next, it uses an approach developed by Yang (1997) to suggest a potential resolution of the identified conflicts. Finally, the CSTN is again used to check whether the proposed resolution observes all the temporal constraints. We have implemented our approach, and we present preliminary experimental evidence about domain factors that influence its performance.

Introduction

In this paper, we develop an algorithm for merging plans that are represented in a richly expressive language. Specifically, we are concerned with plans that have (i) quantitative temporal constraints, (ii) actions that are not instantaneous, but rather have temporal extent, and (iii) conditional branches. Given a set $\mathcal{S}$ of such plans, our algorithm finds a set of constraints that jointly ensure that the plans in S are mutually consistent, if such a set of constraints exists.

Our interest in plan merging is motivated by our work on developing agents for dynamic environments. In these environments, an agent may adopt a goal for a future activity, form a possibly incomplete plan for it, and then consider, commit to, and plan for additional goals before completing—or possibly even beginning—execution of the first plan. Thus, as time proceeds, the agent is continually forming plans for new goals in the context of its existing goals and plans.

Various approaches could be used to generating plans in the context of prior plans. A simple approach would rely on traditional AI methods of planning for conjunctive goals. With this approach, whenever the agent encountered a new goal G in a setting in which it already held goals $G_1, \ldots G_n$, it would form a new planning problem for the conjoined goal $G_1 \wedge \ldots G_n \wedge G$. While straightforward, this approach has at least two serious problems.

- It is computationally inefficient.[1] With each new goal, all the previous planning is thrown out, and a new plan is generated from scratch.
- It may result in highly unstable plans. With complete replanning, the agent never fully commits to its plans, but only to its top-level goals: all of its planned activities are constantly up for reconsideration. The resulting instability may be particularly harmful in multi-agent domains, in which stability of plans is requisite for coordination: a change in one agent's commitments may propagate to another agent, who must then make changes that impact a third agent, and so on.

An alternative approach that avoids these problems is to hold fixed the plans for $G_1, \ldots, G_n$, and merge into them a new plan for G. The plan for G might be generated from scratch, or it might result from the completion of a skeletal plan, as, for example, in case-based planning (Hammond 1989; Kambhampati & Hendler 1992; Hanks & Weld 1992; Veloso & Carbonell 1994). Our focus in this paper is on the process by which the new plan can be merged into a set of existing plans, so that the resulting set is consistent.

The most well-developed prior work on plan merging is that of Yang (1997, Chap. 7), whose algorithms apply to classical plans, i.e., plans expressed using the STRIPS representation.[2] Yang's central idea is to em-

[1] In the section on "Related Work", we discuss the relationship between this claim and apparently related results (Nebel & Koehler 1995).

[2] Yang's book develops several important algorithms. What we describe here as "plan merging" is what he calls "global conflict resolution": it is the process of finding constraints that guarantee that two plans do not negatively

ploy constraint satisfaction processing, modeling the conflicts between plans—the threats—with constrained variables whose domains are their possible resolutions. A solution to this CSP is then a set of constraints that jointly resolve all the threats. We employ Yang's approach as the second of a three-phase process for merging more expressive plans. In the first phase, we identify conflicts between the plans that are to be merged, using a new data structure, conditional simple temporal networks (CSTNs); CSTNs are an extension of STNs (Meiri 1992). In the second phase, we use Yang's method to suggest a potential resolution to the identified conflicts. Finally, the CSTN is again used to check whether the proposed resolution observes all the temporal constraints. We have implemented the algorithm and conducted experiments to evaluate it, and include preliminary results in this paper.

Plan Representation

We represent plans as tuples of the form $P =< S, T, L, A >$, with the components defined as follows[3]:

- S are the *plan steps*. As usual in the planning literature, each step is associated with a particular operator, which has preconditions, effects, and resources. In contrast to the standard approaches, we associate two time points with each step, one representing its start and another representing its end, and we assume functions *Start* and *End* that identify these.
- T is a set of temporal constraints. Elements of T are all of the form: $f(s_i) - g(s_j) \leq d$ where $s_i, s_j \in S$, f and g are either *start* or *end*, and d is a real number. Temporal reasoning problems that are restricted to constraints of this form can be represented with *Simple Temporal Networks* (STNs), and it has been shown that they can be solved in polynomial time, as opposed to more general temporal constraint satisfaction problems which are NP-complete (Meiri 1992). The crucial difference between STNs and general TCPs is that STNs do not allow disjunctive constraints. With the language of STNs, we can still represent the standard ordering constraints of traditional planning systems ("s_i precedes s_j"). We can also represent a wide range of other constraints, as illustrated in Figure 1. To express explicit time assignments (e.g., "s_i begins at 9a.m. Monday") we make use of an initial *reference point* denoting an actual clock time (see the second example in the figure).
- L is a set of causal links, defined in the usual way: $< s_i, e, s_j >$, where e is both an effect of s_i and a precondition of s_j.
- A is a set of alternative-context links (AC links). In general, a conditional plan may include branch points, which separate some subsequent steps in the plan into distinct contexts of execution. We model each branch point with a dummy step of type `branch`, which has outgoing AC links. An AC link $< s_i, r, s_j >$ indicates that if condition r is true *at the time of the branch step* s_i, then step s_j should be executed. The condition r specified on any AC link will be a conjunction of literals. We will assume that the set of AC links emanating from any `branch` step will represent an exhaustive and mutually exclusive set of conditions; this assumption is consistent with the prior literature on conditional planning (Peot & Smith 1992; Pryor & Collins 1996; Onder & Pollack 1997). Note that in this prior literature, branch points are typically equated with observation actions. However, we do not want to restrict our focus to cases in which the observation of a condition immediately precedes the decision about which actions to perform.

To properly handle conditional plans, we need to label each step with the contexts in which it will be executed. Again following the literature on conditional plans, we label steps by propagating AC-link conditions through the plan. We say that two steps occur in *consistent contexts* whenever it is possible for both of them to occur during a single execution of the plan.

An example is given in Figure 2, which represents a plan to go to a meeting that will begin in 60 time units. It is not good to get to the meeting too early, nor to get there late: thus there is a temporal constraint requiring that the agent arrive at the meeting at most 5 time units early. Depending upon the weather, the agent may either walk or drive. If it is sunny (S in the figure), she will walk, and it will take exactly 30 time units. Otherwise, she will drive, taking exactly 10 time units. In either case, the agent must forward her phone calls to her secretary, no more than 1 time unit before departing. Context labels are shown in italics below a step: for instance, the context label for "drive" is $\neg S$. We stress that what matters is whether it is sunny *at the time of the branch step*. If the example plan were to be extended, there might be another, later branch point that also depended upon the condition "sunny"; however, it would refer to the weather conditions at that later time. For clarity, in our examples we will never reuse the same propositional letter, but will instead use different letters to denote, for example, "sunny at time 1" and "sunny at time 2".

Finally, we define a (Type 1) Conflict as follows:

Definition 1 (Type 1 Conflict) A Type 1 Conflict exists between steps s_p and s_t in plan $P =< S, T, L, A >$ if and only if:

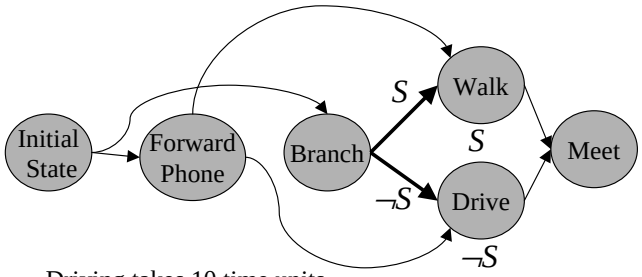
1. $< s_p, e, s_u > \in L$.
2. Step s_t has an effect $\neg e$.
3. Steps s_p and s_t occur in consistent contexts.

interact with one another. Yang reserves the term "plan merging" to refer to the process of combining two or more *steps* of the same type. We prefer to refer to this latter process as "step merging".

[3]For ease of presentation, we focus in this paper on propositional plans. However, our approach can be directly extended to handle plans containing parameterized predicates.

To Express:	Use:
s_i precedes s_j	$end(s_i) - start(s_j) \leq 0$
s_i begins at 9am Mon.	$ref - start(s_i) \leq -9 \wedge start(s_i) - ref \leq 9$ assuming ref is 12am Mon.
s_i lasts between 2 and 3 hours	$end(s_i) - start(s_i) \leq 3 \wedge start(s_i) - end(s_i) \leq -2$
s_i occurs more than 48 hours before s_j	$end(s_i) - start(s_j) \leq -48$

Figure 1: Temporal Constraint Examples



Driving takes 10 time units.
Walking takes 30 time units.

Forwarding the phone has to be done at most 1 minute before walking or driving.

The meeting occurs in 60 time units from the initial state.

We must not arrive more than 5 time units early.

Figure 2: Example of a Conditional Plan

4. Step s_t might begin before step s_u ends, i.e., $start(s_t) - end(s_u) \leq 0$ is consistent with T.
5. Step s_t might end after step s_p begins, i.e., $start(s_p) - end(s_t) \leq 0$ is consistent with T.

This definition generalizes the notion of threats in classical plans, in which clauses 3, 4, and 5 would be replaced by a simpler requirement that s_t may possibly come between s_p and s_u. In this paper, we make the conservative assumption that an intended effect of an action—the effect labeling a causal link—begins at the start of the producing step s_p and must persist to the end of the consuming step s_u. Other situations could be handled by modifying requirements 4 and 5. A similar definition of conflicts appears in the IxTeT system (Ghallab & Laruelle 1994); for further discussion of IxTeT, see the section "Related Research".

By including steps with extended duration, we can also model constraints on conditions that occur during action performance. Each step s in a plan may have an associated set of *resources*, and no other step may "use" one of those resources while s is executing. (That is, if s_1 and s_2 have overlapping resources, then their times of execution may not overlap.) Our treatment of resource constraints derives from the approach originally developed for the Time Map temporal database (Dean & McDermott 1987), and used in systems such as HSTS (Muscettola 1994). The use of resources has turned out to be important for many of the actions we have modeled in an intelligent workflow/calendar management system we are building (Pollack, Tsamardinos, & Horty 1999). For example, in modeling the action of "having a meeting", it is important to record the fact that the agent is busy throughout the duration of the meeting, and therefore is prohibited from scheduling another meeting at the same time. This can readily be done by attaching a resource—the attention of the person attending the meeting—to the "have a meeting" operator.

The inclusion of resources requires a generalization of the definition of conflict. We thus also define Type 2 Conflicts.

Definition 2 (Type 2 Conflict) A Type 2 Conflict exists between two steps s_i and s_j if and only if:

1. Step s_i and s_j both have the same resource c_i.
2. Steps s_i and s_j occur in consistent contexts.
3. Steps s_i and s_j might overlap, i.e., $start(s_i) - end(s_j) \leq 0$ and $start(s_j) - end(s_i) \leq 0$ are both consistent with T.

Our plans are inherently parallel: all steps that are not constrained to occur at different times may be executed simultaneously. The use of resources makes it straightforward to prohibit co-occurrence of steps that would have harmful interactions with one another.

The Plan Merging Algorithm

Our goal is to take two plans expressed in the representation language described above, and find temporal constraints that ensure their consistency. Let $P_1 =< S_1, T_1, L_1, A_1 >$, and $P_2 =< S_2, T_2, L_2, A_2 >$. As described in the introduction, in a dynamic environment P_1 will represent the conjunction of the agent's existing plans, and P_2 will represent its plan for a new goal. Let P be the piecewise union of P_1 and P_2, i.e., $P =< P_1 \cup P_2, T_1 \cup T_2, L_1 \cup L_2, A_1 \cup A_2 >$. Then we seek to find a set of temporal constraints T' that (1) are consistent with $T_1 \cup T_2$ 2) are such that $P' =< P_1 \cup P_2, T_1 \cup T_2 \cup T', L_1 \cup L_2, A_1 \cup A_2 >$ is conflict-free.

As mentioned above, this question was addressed for the case of classical plans by Yang(1997). Given a plan P' that is the piecewise union of two other plans, Yang first identifies its conflicts as defined in the classical framework. Such conflicts can be computed by considering each pair of steps in P', checking the effects of one against the causal links emanating from the other. After identifying all the conflicts, Yang constructs a constraint satisfaction problem (CSP) in which the nodes represent all the threats in P', and the domain of each node is the set of its possible resolutions—promotion, demotion, and separation, as standardly defined. We will call this a *conflict-resolution CSP*. Efficient constraint processing techniques can then be applied to find a solution, which represents a way of jointly resolving all the conflicts in the plan.

Although Yang's approach works well for classical plans, it is not adequate on its own for the more expressive plans with which we are concerned in this paper. First, the conflict identification technique for classical plans is not sufficient for plans with rich temporal constraints. Second, a solution to a Yang-style conflict-resolution CSP will not necessarily respect all the temporal constraints of an expressive plan. For example, consider the plan segment in Figure 3. Both step S_l and step S_j conflict with step S_k, because S_k has effect $\neg p$. Because Yang's algorithm ignores temporal information, it could produce a solution in which S_k is promoted to occur after S_i and also demoted to occur before S_j. This solution is consistent if we ignore the quantitative temporal constraints. However, our plan in fact has temporal constraints: the duration of S_k is at least 20 time units, and S_j must start at most 10 time units after the end of S_i. Then the proposed solution is obviously not valid: it is not consistent with these temporal constraints.

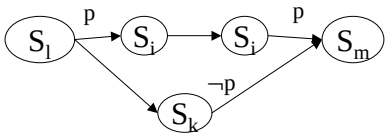


Figure 3: Example of the need for the STN

To address these problems, we have developed a three-phase plan merging algorithm, shown in Figure 4. At initialization (Step 1), a plan structure is created by performing a piecewise union of the components of the existing plan and the new plan to be merged in. In the first major processing phase (Steps 2-3), we rely on a new data structure, Conditional Simple Temporal Networks (CSTNs), defined below. A CSTN is constructed and then used to identify all the conflicts in the plan. In the second phase (Steps 4-5), a conflict-resolution CSP is constructed to represent the conflicts found in Step 3. Constraint satisfaction techniques are applied to find a solution to it. Finally (Steps 6-8), the CSTN is used again, to check whether the proposed solution respects all the plan's temporal constraints. If it does not, then backtracking in the conflict-resolution CSP is performed and an alternative potential solution is produced; this process is repeated until either a solution is found that is consistent with the temporal constraints, or no more solutions to the conflict-resolution CSP exist. We next provide details of each of the three main phases of the algorithm.

CSTN Construction and Conflict Identification

To support reasoning about the temporal constraints in our plans, we make use of a new data structure, a Conditional Simple Temporal Network (CSTN). CSTNs ex-

Merge-Plans($P1$, $P2$)
1. Let $P = P1 \cup P2$.
2. Construct a CSTN representing P.
3. Use the CSTN to identify all conflicts in P.
4. Construct a conflict-resolution CSP encoding the identified conflicts.
5. Find a solution, $\mathcal{C}$, to the conflict-resolution CSP. (Set backtrack point.) If there are no more solutions, return failure.
6. Add the constraints in $\mathcal{C}$ to the CSTN and check to determine whether it is consistent.
7. If the updated CSTN is consistent, then return $\mathcal{C}$,
8. Else, restore the CSTN to its original state, and backtrack at line 5.

Figure 4: The Top-Level Algorithm

tend Simple Temporal Networks, a well-known data structure from the temporal-reasoning literature. An STN is a graph consisting of nodes that represent events, and arcs that represent constraints on the time intervals between the events. As mentioned earlier, each constraint in an STN has the form:

$$X_i - X_j \leq d$$

indicating that event associated with node X_i must occur no later than d units of time after the event associated with node X_j. The value d is called the *weight* of the arc.

STNs are not expressive enough for our purposes. Although we could use them to model and reason about plans with rich temporal constraints, they do not support reasoning about conditional branches. We therefore extend STNs, by importing the idea of context labels from conditional plans. Specifically, we define a CSTN to be an STN in which each node has an associated set of context labels.

We can construct a CSTN from a plan using the algorithm in Figure 5. The idea is straightforward: the nodes in the CSTN correspond to the start and end points of the steps in the plan, and the arcs in the CSTN represent both explicit temporal constraints and implicit ones that are derived from causal and AC links. The derived CSTN for the example of Figure 2 is shown in Figure 6[4].

Having constructed the CSTN for a given plan, we next use it, along with the original plan itself, to identify the conflicts using two definitions given above. We use the plan to find steps that may potentially be in conflict with one another and we use the CSTN to check whether the temporal constraints permit the affected steps to overlap. To determine whether two steps overlap, we have to construct a distance graph for the CSTN

[4]To simplify the figure, we adopt the standard STN approach of displaying a single arc from X_i to X_j labeled $[l, u]$ as an abbreviation for the two arcs associated with $X_j - X_i \leq u$ and $X_i - X_j \leq -l$.

1. Given a plan $P =< S, T, L, A >$
2. For each step s_i in S, create two nodes in the CSTN, one for the start of s_i and the other for its end.
3. Label each node in the CSTN with the context label from its corresponding step in P. For each causal and AC link $< s_i, r, s_j >$ in L, add an arc from $start(s_j)$ to $end(s_i)$ with weight 0.
4. For each temporal constraint $f(s_i) - g(s_j) \leq d$, add an arc from $g(s_j)$ to $f(s_i)$ with weight d.

Figure 5: Constructing a CSTN from a Plan

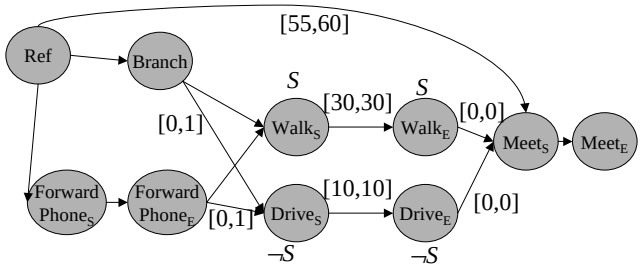


Figure 6: Derived CSTN for the Plan in Figure 2.

as explained below in the section on Solution Validation.

Conflict-Resolution CSP Construction and Candidate Solution Generation

Having identified all the conflicts between the plans, our algorithm enters its second phase. Here we follow Yang's approach directly: we construct a CSP with a node for each conflict and set the domain of each node to be the set of possible resolutions. Because we are focusing on propositional plans here, this set will contain simply promotion and demotion; in the more general case, it would also contain separation. For example, as already mentioned, the plan segment of Figure 3 contains two conflicts, one between S_l and S_k, and the other between S_j and S_k. The Conflict-Resolution CSP will contain variables $V_{l,k}$ and $V_{j,k}$, corresponding to the two conflicts with domains $D_{l,k} = \{\{S_k < S_l\}, \{S_i < S_k\}\}$, and $D_{j,k} = \{\{S_k < S_j\}, \{S_m < S_k\}\}$ respectively.

We then extend Yang's definition of consistency between variable assignments in the conflict-resolution CSP, to take into consideration the quantitative temporal constraints and the conditions. Let d be in the domain of some constrained variable V_i and let f be in the domain of another variable V_j, both variables in the conflict-CSP for a plan P. Let C be the CSTN for a plan P as it is output by the algorithm at Figure 5. Then $V_i = d$ is *inconsistent* with $V_j = f$ if and only if the CSTN $C \cup d \cup f$ is inconsistent. The assignments $V_i = d$ and $V_j = f$ are consistent if and only if they are

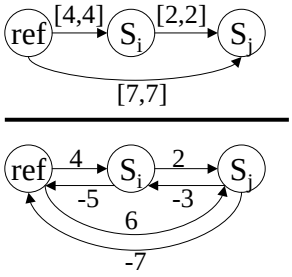


Figure 7: An Inconsistent STN and its Distance Graph

not inconsistent. Informally, a CSTN is inconsistent if there is no assignment of values to variables that satisfies all the constraints; a more precise definition is given in a later section.

Again illustrating with Figure 3 and the two conflicts $V_{l,k}$ and $V_{j,k}$ that it introduces, the following assignment is inconsistent $< V_{l,k} = \{S_k < S_l\}, V_{j,k} = \{S_m < S_k\} >$, as is $< V_{l,k} = \{S_i < S_k\}, V_{j,k} = \{S_k < S_j\} >$. The latter assignment is inconsistent because of the quantitative constraints on the duration of step S_k and the maximum temporal separation of steps S_i and S_k.

Efficient constraint processing techniques can be used to search for a solution to the conflict-resolution CSP. For instance, we can compute arc-consistency and/or k-consistency, followed by search in the space of variable assignments. Optimization techniques from the CSP literature, such as forward checking, backjumping, and so on, can also be applied. Yang also describes a problem-specific subsumption relation that can be used to prune variables and values from the search.

This second phase of our main algorithm terminates with a candidate solution to the plan merging problem. But as we observed in Figure 3, this candidate may not actually be valid. Thus a third phase of processing is required.

Solution Validation

To check the consistency of the proposed solution with the full set of temporal constraints, we augment the CSTN built in step 2 of the main algorithm with the temporal constraints from the solution proposed in step 5. We then check to see if the resulting CSTN is consistent.

CSTN consistency is modeled on STN consistency. An STN is consistent if and only if it is possible to assign a time point to each event represented in the STN while guaranteeing that all the temporal constraints are satisfied. The top part of Figure 7 illustrates an inconsistent STN. To respect the constraints, s_i would have to occur exactly four time units after the reference point (i.e., "at time 4"), s_j would have to occur exactly two time units later (i.e., at time 6), but s_j would also have to occur at exactly time 7). In contrast, if the time between s_i and s_j was constrained to be between 1 and 3 time units, the STN would be consistent.

STN consistency can be computed using shortest-path algorithms such as Bellman-Ford (Cormen, Leiserson, & Rivest 1990), which are known to be relatively efficient (i.e. polynomial time in the number of nodes in the STN). For any STN we can construct a *distance graph* in which the arc from X_i to X_j is labeled by the length of the shortest path from X_i to X_j in the original STN. It has been shown (Meiri 1992) that:

- An STN is consistent iff its distance graph contains no negative cycles, and
- if the arc from X_i to X_j in the distance graph has label d, then $X_i - X_j \leq d$ is entailed by the original STN; moreover, d is the smallest such number for which this is true.

Note the negative cycle between Ref and s_i in the bottom part of Figure 7, which is the distance graph [5] for the STN above it.

CSTN Consistency The notion of consistency in a CSTN is not as straightforward. Intuitively, we would like to use a very similar definition, and say that a CSTN is consistent precisely when we can assign time points to the represented events in such a way that all constraints are respected. However, each event in a CSTN has an associated context label that specifies the contexts in which it will be executed. Events in the same CSTN may occur in inconsistent contexts. Recall, for example, our meeting example: walking only occurs when the weather is sunny at the time of the first branch action, while driving occurs only when the weather is not sunny at that time. There are a finite number n of distinct *executions*—or "ways the plan might go"—depending on the conditions that occur. For the library example, there are only two executions. The maximum number of possible executions is no greater than 2^c, where c is the number of propositional symbols used to describe conditions.

We can identify several different notions of consistency with which one might be concerned. A CSTN will be consistent in a very strong sense (*strong consistency*) if there exists an assignment of time points to events in the CSTN such that the temporal constraints will be respected regardless of the context in which the represented plan is actually executed. Strong consistency is a highly desirable property for a plan to have.

Unfortunately, many times the proposed solution to a plan merging problem will not be strongly consistent. Assume that our meeting plan in Figure 2 (with derived CSTN in Figure 6) is part of a solution to a merging problem. At the time that the agent is considering this solution, she does not know what the weather will be like at the appropriate time in her plan. Indeed, her meeting may not even occur until a few days after she adopts the plan for it. Yet, as the reader can determine from Figure 6, the agent needs to forward her phone calls between time points 24 and 25 (relative to the reference point) if she is going to walk to the meeting, but needs to forward the phone calls between time points 44 and 45 if she is going to drive. Thus for this example there is no single assignment of times to events that will guarantee that all the temporal constraints are satisfied regardless of the context of execution. However, if the agent is able to determine the execution context before plan execution begins, then she will be able to make a satisfactory temporal assignment to all the steps. In a situation like this, we will say that the plan is *weakly consistent*: for each possible execution, there exists an assignment of time points to events that respects the temporal constraints, but different possible executions may require different time assignments. In our example, forwarding the phones can be assigned time 24 for the sunny context of execution, and time 44 for the opposite case.

A range of intermediate cases occur between strong and weak consistency, having to do with situations in which the agent determines the context of execution incrementally during execution itself. Very informally, a plan will be *dynamically consistent* if and only if all the information is learned "soon enough" to ensure time assignments can be made such that all temporal constraints will end up being respected.

Strong, weak, and dynamic consistency, as we have defined them, are related to the identically named notions in Vidal and Ghallab's work (Vidal & Ghallab 1996). However, Vidal and Ghallab model a different sort of uncertainty than we do. They define a special type of STNs in which the time of occurrence of some events is outside the control of the agent. In contrast, in our approach, all events are assumed controllable, but there is uncertainty about which events will actually be executed. Neither approach subsumes the other: in future work it would be useful to combine the two.

Checking CSTN Consistency In step 6 of our main algorithm (Figure 4), we could check for whichever type of CSTN consistency is most appropriate for the current situation. To check for strong consistency, we simply ignore the context labels altogether, and treat the CSTN as if it were an STN. After all, what we are looking for in this case is a single assignment of times to points that "works" regardless of what context the plan ends up being executed in. We therefore directly run the Bellman-Ford algorithm, and check that there are no negative cycles. The complexity of this algorithm is well known to be $O(ve)$, or equivalently $O(v^3)$ for dense graphs, where v is the number of nodes in the CSTN and e the number of edges or constraints.

The process of checking for weak consistency is more interesting. In this case, rather than checking whether there exists an assignment that observes *all* the temporal constraints simultaneously, we need to consider each possible execution, and determine whether there is a satisfactory time assignment for it. The brute force

[5] Technically, when there is a negative cycle all shortest paths go to negative infinity. We show what would be the output of the Floyd-Warshall algorithm (Cormen, Leiserson, & Rivest 1990)

approach to this involves running the Bellman-Ford algorithm once for each possible execution. Each time we take into account only those nodes whose context labels are consistent with the particular execution context, and we similarly only consider the constraints amongst those nodes. For each possible execution, we then make sure that the resulting distance graph has no negative cycles. The complexity of checking for weak consistency is clearly $O(v^3) * (2^c))$, where again, c is the size of the set of propositional labels for describing conditions.

Checking for dynamic consistency is more complex, and we defer discussion of that to another paper.

An Alternative Approach

Our algorithm makes use of two constraint satisfaction networks. In one—the conflict-resolution CSP—the nodes represent conflicts and their domains represent possible resolutions of those conflicts. The constraints between nodes are consistency constraints, which encode the conditions under which conflict resolution methods are incompatible with one another. In the other—a CSTN—the nodes represent events (the start time and end time of steps in the plan) and their domains represent times for those events. The constraints represent the durations of events and the intervals of time between them.

It is reasonable to ask whether we could simplify the process to make use of a single CSP. For the moment, assume that plans do not have branches, and that, as in the current paper, separation is not available as a conflict resolution method. Even then we would need to make use of general temporal constraint networks (Meiri 1992) to capture both quantitative temporal constraints between time points, and the disjunctive qualitative constraints that are necessary to encode the alternatives of promotion and demotion. Also, once we allow for separation, we will require a larger set of nodes in the CSP: they cannot represent only time points, because separation involves a binding constraint, not a temporal constraint. Suppose that we had a heterogeneous CSP, in which some nodes represent time points and other nodes represent domain parameters. Although this is possible, it would involve further several complications: we could not restrict the constraints to be binary (they would have to involve several nodes); we could not directly rely on shortest-path algorithms, since the CSP would not have only temporal nodes; and we would still need to introduce techniques to model branching. Given these complexities, we believe it is more reasonable to use two different types of CSPs at different stages in the processing. However, this does imply that the only possibility is to *completely* solve the conflict-CSP prior to checking its validity with the CSTN. We are currently investigating related designs, in which partial solutions to the conflict-CSPs are verified incrementally.

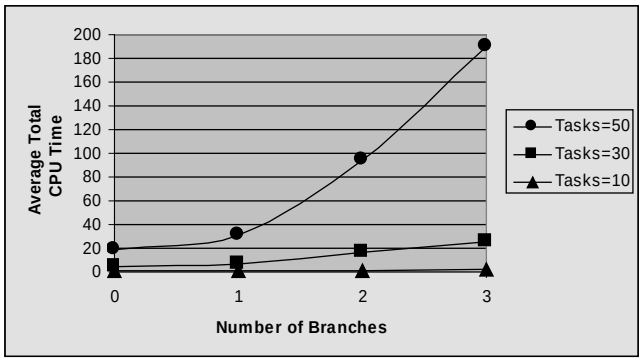


Figure 8: Average Total Time to Merge Plans

Preliminary Experimental Results

We implemented the plan merging algorithm described above in Allegro Common Lisp on a Pentium machine running Windows-NT. We then generated a large number of plans over a set of artificial step types. (Note that these plans are meant to represent the piecewise union of two plans that are to be merged, i.e., we bypass step 1 of our main algorithm.) We initially generated random skeletal plans (i.e., sets of steps), and then randomly added preconditions, effects, causal links, and temporal constraints with condition labels coming from a fixed set of propositional symbols. Conflicts were not directly introduced by us, but rather arose as a side-effect of the random generation process. During generation, we varied a number of factors, including the number of tasks (i.e. steps), the number of branches and the maximum time span allowed for the plan. In total, we ran the algorithm on nearly 2000 distinct problems. Our goal in conducting these experiments was to gain an understanding of the domain influences that affect performance. Here we present some preliminary results from this effort. These results should be viewed as suggestive of the algorithm's performance and of likely places for investigating efficiency gains. We will present a more thorough and systematic analysis of the results in another paper.

Figure 8 shows the overall time taken by our algorithm to either find a solution to a plan merging problem or determine that no such solution exists. In all our experiments, we checked for the more challenging notion of weak consistency. We plot separate lines for the average time taken to merge plans with a total of 10, 30, and 50 steps, in each case varying the number of branches from 0 to 3.

The key factor in the overall time taken appears to be the number of times that the CSTN must be called to check potential solutions proposed by the conflict-resolution CSP. This in turn depends heavily on how "tight" the temporal constraints are. When the maximum time span allowed for the merged plan is smaller, the constraints are tighter, and it is more likely that a potential solution will violate a temporal constraint. Figure 9 shows the mean and maximum number of calls

Maximum Time Span	Mean # of Candidate Solutions	Maximum # of Candidate Solutions
210	0.523364486	1
180	0.648148148	1
150	0.691588785	7
120	1.696202532	39
90	76.97163121	1872

Figure 9: Influence of Tightness of Constraints on Amount of Backtracking

to the CSTN (Step 6 in the algorithm) for plan merging problems with a total of 30 steps, each of which had average duration of 10 time units, for various time spans. The mean can be less than one because in some cases the process of constructing the original CSTN (Step 2) is sufficient to determine that the plans cannot be successfully merged. Recall that non-conflicting steps can be performed in parallel or can overlap, which is why 30 steps can be scheduled in as little as 90 time units.

Solving the conflict-resolution CSP and the CSTN are both exponentially hard problems. Interestingly though, in practice solving the conflict-resolution CSP took very little time on average, usually requiring no more than 20% of the overall processing time. Thus, we are currently focusing our attention on optimizing the procedure for solving the CSTN.

Related Research

This paper aims at developing a theory of plan merging for agents in dynamic environments. We argued in the introduction that as new goals arise for such agents, it is better to merge a plan for the new goal into the agent's existing set of plans, rather than to replan for the conjunction of the existing goals and the new one. In this regard, we must mention the work of Nebel and Koehler (Nebel & Koehler 1995), which showed analytically that plan reuse cannot lead to a provable worst-case efficiency gain over plan generation; indeed, plan reuse can be strictly worse than plan generation.[6]

Given these results, one might question our decision to pursue plan merging. However, the Nebel and Koehler analysis does not directly apply to the situation we are modeling. Their analysis assumes that the planning agent has a library of plans $\Lambda = P(G_1), \ldots, P(G_n)$ for some individual goals, $G_1, \ldots, G_n$. When the agent then has to form a plan for a new goal, G, the plan reuse approach is to find a plan $P(G_i)$ in Λ for a goal G_i that is appropriately similar to the new goal G, and then to modify $P(G_i)$ so that it can serve as a plan for G. Nebel and Koehler show that the bottleneck in this approach is the retrieval of the appropriate plan $P(G_i)$ to modify.

The differences in the situation which we are concerned are significant. In our setting the agent does not simply have a library of plans $\Lambda = P(G_1), \ldots, (G_n)$; rather the agent is *committed* to $P(G_1) \wedge \ldots P(G_n)$, as a result of decisions it has made in the past. When it encounters a new goal G, its job is to add to its set of commitments a new plan to achieve G. It is unlikely that the agent will even have stored away a plan for a goal that is "similar to" the complete conjunction of goals $G_1 \wedge \ldots \wedge G_n \wedge G$. After all, this would require the agent to have plans for all combinations of goals it might encounter. The agent in our dynamic setting does not search a plan library to find a plan for a goal that is similar to its new, conjunctive goal. Rather it makes use of a single plan, the one to which it is already committed, and then merges into that a plan for the new goal.

Of course, this does not conclusively prove that plan merging will be computationally more effective than replanning from scratch in such settings, although we suspect that it will be, given the fact that the large conjunctive plan that addresses the existing goals will tend to contain very large segments that will in no way interfere with the new plan. However, even if our hypothesis about computation time is not borne out, the added stability that results from plan merging in our setting is quite important. Nebel and Koehler also recognize the importance of stability, noting that "in a *replanning context*, [in which] a plan has to be modified because of user-initiated specification changes or execution failures, one may want to respect as many previous commitments as possible" (Nebel & Koehler 1995, p.432).

A number of other researchers have studied the problems involved in plan *generation* under temporal constraints, for example including (Bacchus & Kabanza 1996; Ghallab & Laruelle 1994; Currie & Tate 1991; Dean & McDermott 1987; Vere 1983). Amongst this literature, the work that is most similar to our own is that on the IxTeT system by Ghallab and his colleagues. IxTeT has a rich language for expressing temporal constraints in plans, and, along with several other planning systems, separates reasoning about temporal constraints from reasoning about binding, resource and other types of constraints. The IxTeT system uses a slightly different approach to temporal reasoning, relying on the restricted interval algebra. However, this algebra precisely precludes disjunctive constraints, as do STNs, and the reasoning algorithms used in IxTeT, along with their definition of conflicts and consistency, are similar to our own. Our work differs from that of IxTeT in two key ways. First, we introduce the notion of context to handle conditional plans. Conditional plans are common, and, as we noted above, they raise a host of interesting questions that we are continuing to pursue. Second, IxTeT is performing plan generation, using a fairly standard partial-order planning algorithm; thus, it analyzes threats one at a time. In contrast, we

[6] Nebel and Koehler also provide empirical evidence from the blocks world showing that actual performance times tend to observe this same pattern.

are making use of Yang's innovative idea of reasoning about all the threats in a plan at a single time. We are able to do this because we are performing plan merging, and thus have complete plans to begin with. Prior work by Joslin and Pollack (Joslin & Pollack 1996) explored an intermediate approach that allows one to generate plans in the classical way, while still reasoning about all the conflicts that exist at any given point in the planning process.

Conclusion

We are concerned with developing agents for dynamic environments, who must adopt plans for new goals in the context of their existing plans. In this paper, we have presented an algorithm for merging plans with conditional branches and rich temporal constraints. As part of this work, we developed Conditional Simple Temporal Networks, a new representation for expressing quantitative temporal constraints on events that may occur only in specificied contexts.

We are currently extending the work along several dimensions. First, we are extending our analysis of the performance data to more accurately pinpoint the domain influences on efficiency. Second, we are developing a range of heuristics for improving the efficiency of checking the consistency of a CSTN. In particular, we are exploring the reuse of partial solutions in different execution contexts. Third, we are studying dynamic consistency, mentioned in the section "Solution Validation", and are developing techniques for determining when observations about execution context need to be made. Finally, we are using this algorithm in a project to develop an intelligent calendar manager, and will be investigating the potential for using it for a variety of workflow tasks.

Acknowledgments

This research was partially supported by the Air Force Office of Scientific Research (F49620-98-1-0436), by the National Science Foundation (IRI-9619579), and by a University of Pittsburgh Andrew Mellon Predoctoral Fellowship.

References

Bacchus, F., and Kabanza, F. 1996. Planning for temporally extended goals. In *Proceedings of the 13th National Conference on Artificial Intelligence.*

Cormen, T. H.; Leiserson, C. E.; and Rivest, R. L. 1990. *Introduction to Algorithms.* Cambridge, MA: MIT Press.

Currie, K., and Tate, A. 1991. O-plan: The open planning architecture. *Artificial Intelligence* 52:49–86.

Dean, T. L., and McDermott, D. 1987. Temporal data base management. *Artificial Intelligence* 32:1–55.

Ghallab, M., and Laruelle, H. 1994. Representation and control in IxTeT, a temporal planner. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS-94)*, 61–67.

Hammond, K. J. 1989. *Case-Based Planning: Viewing Planning as a Memory Task.* New York: Academic Press.

Hanks, S., and Weld, D. S. 1992. Systematic adaptation for case-based planning. In *Proceedings of the First International Confernce on AI Planning Systems*, 96–105. Morgan Kaufmann.

Joslin, D., and Pollack, M. E. 1996. Is 'early commitment' in plan generation ever a good idea? In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI)*, 1188–1193.

Kambhampati, S., and Hendler, J. 1992. A validation structure based theory of plan modification and reuse. *Artificial Intelligence* 55(2):193–158.

Meiri, I. 1992. *Temporal Reasoning: A Constraint-Based Approach.* Ph.D. Dissertation, UCLA.

Muscettola, N. 1994. HSTS: Integrating planning and scheduling. In Zweben, M., and Fox, M. S., eds., *Intelligent Scheduling.* San Francisco: Morgan Kaufman. 169–212.

Nebel, B., and Koehler, J. 1995. Plan reuse versus plan generation: A theoretical and empirical analysis. *Artificial Intelligence* 76:427–454.

Onder, N., and Pollack, M. E. 1997. Contingency selection in plan generation. In *Proceedings of the 4th European Conference on Planning*, 364–376.

Peot, M., and Smith, D. E. 1992. Conditional nonlinear planning. In *Proceedings of the First International Conference on AI Planning Systems (AIPS-92)*, 189–197.

Pollack, M. E.; Tsamardinos, I.; and Horty, J. F. 1999. Adjustable autonomy for a plan management agent. In *1999 AAAI Spring Symposium on Agents with Adjustable Autonomy.*

Pryor, L., and Collins, G. 1996. Planning for contingencies: A decision-based approach. *Journal of Artificial Intelligence Research* 4:287–339.

Veloso, M. M., and Carbonell, J. G. 1994. Case-based reasoning in prodigy. In Michalski, R. S., and Teccuci, G., eds., *Machine Learning: A Multistrategy Approach, Volume IV.* Morgan Kaufmann. 523–548.

Vere, S. 1983. Planning in time: Windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 5(3):246–267.

Vidal, T., and Ghallab, M. 1996. Dealing with uncertain durations in temporal constraint networks dedicated to planning. In *The 12th European Conference on Artificial Intelligence*, 48–52.

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