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: $l_1 \leq x_1 - y_1 \leq u_1 \;\vee$

…$\vee$ $l_n \leq x_n - y_n \leq u_n$, 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∨ 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∨ M)∧(L∨ N∨ 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 $t \notin \text{TW}(x,i)$.
3. Propagate the constraint $t \leq \text{x} - \text{TR} \leq t$ in all remaining Solutions.
4. Mark *x* as Executed.
5. Compute-Dispatch-Info (Solutions).

Update-for-Violated-Bounds (STP[i] Solutions)

1. Let $U = \{\text{U}(x, k) | \text{U}(x, k) < \textit{Current-Time}\}$
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. $\text{ET} = \text{ET} \cup \langle x, \text{TW}(x)\rangle\}$.
4. Let U = the set of upper bounds on time windows, $\text{U}(x,i)$ for each still unexecuted 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 $\text{U}_{MIN} = \{\text{U}(x, i) | \text{U}(x,i) = NC\}$.
7. For each *x* such that $\text{U}(x,i) \in \text{U}_{MIN}$, let $S_x = \{i \mid \text{U}(x,i) \in \text{U}_{MIN}\}$
8. {Initialize F = true;
9. For each minimal solution MinCover of the set-cover problem (Solutions, $\cup S_x$), let $F = F \wedge (\vee_{x \mid Sx \in MinCover}\, x)$.
10. $\text{DF} = \langle F, NC\rangle$.}

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 <ET,DF>$_1$ . . . ,<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.