UNIVERSITY OF TORONTO
Department of Computer Science

DEPTH EXAMINATION REPORT

Plan Dispatchability: A Survey

Author:
Christian MUISE

Supervisors:
Prof. Sheila A. MCILRAITH
Prof. J. Christopher BECK

February 10, 2016

Contents

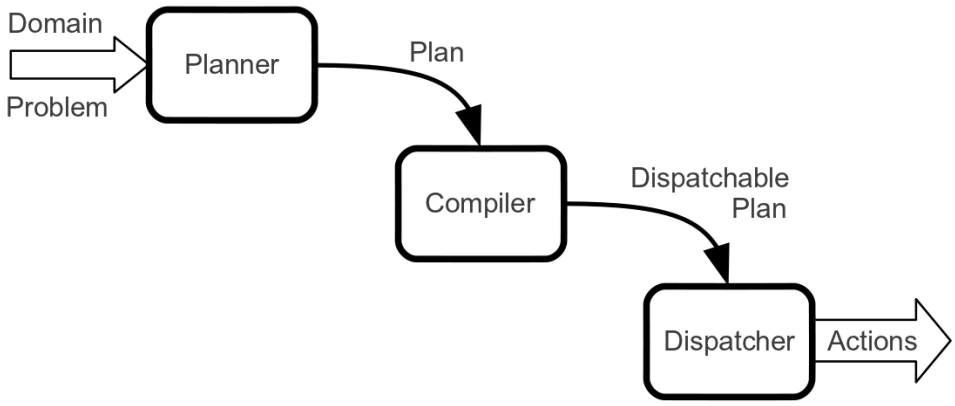


Figure 1: Overview of the general planning and dispatching framework.

1 Introduction

The task of automated planning naturally involves two central issues: how to generate a viable plan for a given goal, and how to properly execute a plan that has been constructed. It is common to treat these issues in isolation, for example plan generation research for the former, and execution monitoring for the latter. However, at times it is important to consider how the two worlds can be bridged. In this review, we consider the area of *plan dispatchability*: the problem of determining if (and how) a plan can (and should) be executed. Figure 1 provides a general framework that demonstrates where plan dispatchability fits in. Here, a plan produced by a planner is processed by a *compiler* phase to produce a dispatchable plan or schedule. A subsequent *dispatcher* then uses the compiled form to choose the appropriate time for executing an action, while doing any online scheduling that is necessary. It is the dispatchable form that allows for effective online execution and scheduling by the dispatcher.

In this paper we focus on dispatching plans that are in the form of a *Simple Temporal Network* (STN) and its variants. A simple temporal network, introduced in the 1980's by Dechter et al. [Dechter et al., 1989], is a form of solution that primarily deals with the ordering of events, and the timing restrictions between them. Durative actions are represented by using a pair of events, and a number of temporal formalisms can be expressed by a combination of temporal constraints in an STN (e.g., many of the rules of Allen's Interval Algebra [Allen, 1984]). Executing the events in an STN is referred to as *dispatching* the network, and a network that can be dispatched properly is called *consistent*. The problem of checking the consistency of a STN is referred to as a the *Simple Temporal Problem*, and remains as an integral part of planning systems today (e.g., checking the continued consistency of a network in temporal planners).

After the introduction of STP, further research branched in two orthogonal directions: adding uncertainty to the temporal constraints and adding richness to the representation language. For the latter, disjunctions of temporal constraints were introduced, and reasoning about the consistency of a network had to be adjusted accordingly. More recently, even richer forms have been introduced that allow for the agent to choose *which* events to execute (rather than simply *when* they should be executed).

To add uncertainty, the temporal constraints were modified so that some of the durations were out of control of the dispatching algorithm – nature would decide the duration of the action. From this viewpoint, three notions of *controllability* were introduced to replace the notion of consistency: *strong*, *weak*, and *dynamic* controllability. At an intuitive level, strong controllability holds when a single solution to the

timing of events works in *any* situation nature chooses; weak controllability holds when there exists a timing of the events for every situation nature chooses; and dynamic controllability holds when we can dynamically choose the timing of events such that the remainder of the network will still be executable (regardless of what nature chooses).

The two research directions in network dispatchability, while orthogonal, are not incompatible. For many of the combinations of network richness and controllability requirement, there has been research that focuses on how to handle the particular case. In this paper we survey many of these approaches to present an overall view of how the problems are handled.

The techniques for plan dispatchability come into play after a plan has been generated, but prior to its execution. While we focus primarily on the temporal aspects surrounding this part of planning, there are parallels for logic formalisms as well as scheduling (e.g., [Wilkins, 1985; Knoblock, 1995; Veloso et al., 1998] and the work to unify scheduling and dispatching under the same general framework [Bidot et al., 2009]). We will address these connections briefly later in the paper.

We begin our review of plan dispatchability by introducing many of the classic formalisms and problem definitions in Section 2. We follow with a discussion of the various approaches for determining consistency and controllability in Section 3. In Section 4 we cover a few extensions to the classic problems that have been considered. We conclude with a brief discussion on the relation to other plan execution techniques in Section 5 and concluding remarks in Section 6. For convenience, we provide a glossary for terms used throughout the paper in Appendix A.

2 Classic Formalisms

In its simplest form, a *Simple Temporal Network* (STN) consists of a set of events that must occur at some point, and a set of temporal constraints amongst the events. We will interchange the terminology "constraint" and "link" between two events, as a link in the graphical representation of a network serves as a representation for some constraint. The various formalisms presented in this section address some of the ways in which the temporal constraints have been extended and the ways in which uncertainty is introduced into the model. We also survey the key problems that are considered in the literature with the various forms of representations.

2.1 Temporal Problems

For temporal networks without uncertainty, the key question to be answered is whether or not the network is *consistent*. Intuitively, a network is consistent if the events in the network can be scheduled such that every constraint is satisfied. The following variations for STNs change the type of constraint that is allowed, progressively adding expressive power to the formalism, but the notion of consistency remains the same. For every type of network we introduce (STN, TCSN, DTN), the problem of determining consistency of the network is simply referred to as the "problem" for that type – e.g., determining consistency of an STN is called the *Simple Temporal Problem* (STP).

Once a network is shown to be consistent, it is also important to describe how it should be dispatched. As we will see later (cf. Section 3), many approaches take the strategy of *least-commitment*: executing actions in such a way so as to maximize the flexibility of the remaining plan. Often, information gleaned while proving consistency can be used for the effective dispatch of the network. This approach will be seen throughout Section 3.1 when we cover the methods for checking consistency and dispatching of a consistent network.

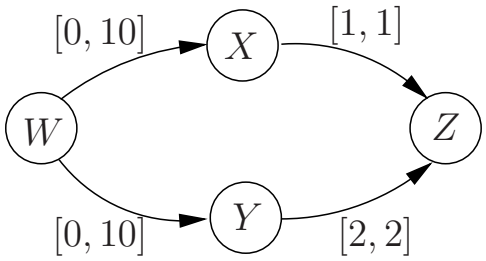


Figure 2: Example STN.

2.1.1 STN

A Simple Temporal Network (STN) consists of a set of instantaneous events, and a set of simple temporal constraints over those events. Every event in an STN must be executed, and we will use capital letters to refer to them (X, Y, etc.) while the symbol T indicates the time an event is executed (T_X, T_Y, etc.). The function T is called a *schedule*, as it assigns a time point to every event, and we use $\mathcal{T}$ to refer to the set of all schedules. A *simple temporal constraint* is a constraint between two events that restricts when the events can occur relative to one another. Formally, we define a simple temporal constraint as follows:

Definition 1 (Simple Temporal Constraint)**.** For events X and Y, $[l, u]_{XY}$ is the simple temporal constraint that restricts the timing of events X and Y to be,

$$l \leq T_Y - T_X \leq u$$

The values of l and u may be any real number (possibly negative, and including $\pm\infty$), but must satisfy $l \leq u$.

Note that the simple temporal constraint does *not* give an indication of the absolute time that an event must occur. Rather, it simply indicates the relative timing between two events. The constraint $[l, u]_{XY}$ can be interpreted as "event Y must occur no earlier than l (and no later than u) time units after X occurs".

A complementary view on the simple temporal constraint will be quite useful throughout the paper: from the equation, $l \leq T_Y - T_X \leq u$, we can infer the constraints $T_Y - T_X \leq u$ and $T_X - T_Y \leq -l$. In both cases, the constraint of the form $A - B \leq b$ can be read as "b is an upper bound on the time A can occur after B occurs".

Example 1 (STN)**.** Figure 2 shows an example of an STN with 4 events (W, X, Y, Z), and 4 simple temporal constraints amongst the events:

- $[l, u]_{WX} = [0, 10]$
- $[l, u]_{WY} = [0, 10]$
- $[l, u]_{XZ} = [1, 1]$
- $[l, u]_{YZ} = [2, 2]$

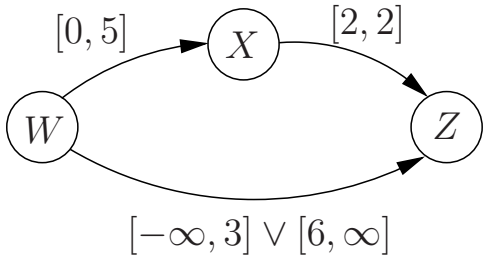


Figure 3: Example TCSN.

We will typically include a starting event for every network (W in Figure 2) to indicate the "start of time". The starting event is assumed to be scheduled at time $t = 0$.

The Simple Temporal Problem (STP) refers to the task of determining if an STN is consistent. One potential (naive) approach is to find a schedule such that all of the constraints are satisfied. As we will see later, there are sufficient and necessary conditions that are easier to compute for ensuring consistency. Section 3.1.1 will further describe how a consistent STN can be dispatched.

Returning to our example, we can see the STN is consistent by using the following setting for the timing of the 4 events: $T_W = 0,\ T_X = 4,\ T_Y = 3,$ and $T_Z = 5$.

2.1.2 TCSN

To increase the expressiveness of STNs, [Dechter et al., 1989] introduced *Temporal Constraint Satisfaction Network* (TCSN). Similar to an STN, a TCSN is made up of a set of events and a set of temporal constraints amongst those events. However, rather than only containing simple temporal constraints, we allow disjunctions of simple temporal constraints for the same two events. Formally, we allow disjunctive constraints of the following form in a TCSN:

Definition 2 ((Simple) Disjunctive Temporal Constraint)**.** For events X and Y, $\{[l_1, u_1], \cdots, [l_n, u_n]\}_{XY}$ is the simple disjunctive temporal constraint that restricts the timing of events X and Y to be,

$$l_1 \leq T_Y - T_X \leq u_1 \vee \cdots \vee l_n \leq T_Y - T_X \leq u_n$$

In other words, the events X and Y need only satisfy one of the simple temporal constraints in the set $\{[l_1, u_1], \cdots, [l_n, u_n]\}_{XY}$. Similar to STNs, the *Temporal Constraint Satisfaction Problem* (TCSP) refers to a setting of the variables T_* such that every constraint is satisfied (i.e., at least one disjunct in every constraint is satisfied).

Example 2 (TCSN)**.** In Figure 3 we give an example TCSN with 3 events (W, X, Z), and three temporal constraints:

- $\{[l_1, u_1], \cdots, [l_n, u_n]\}_{WX} = \{[0, 5]\}$
- $\{[l_1, u_1], \cdots, [l_n, u_n]\}_{XZ} = \{[2, 2]\}$
- $\{[l_1, u_1], \cdots, [l_n, u_n]\}_{WZ} = \{[-\infty, 3], [6, \infty]\}$

Here, the disjunctive constraint between W and Z can be interpreted as the entire length of the execution not being allowed to take between 3 and 6 units of time (not inclusive). This TCSN is consistent, and a solution to the corresponding TCSP is to set $T_W = 0$, $T_X = 5$, and $T_Z = 7$.

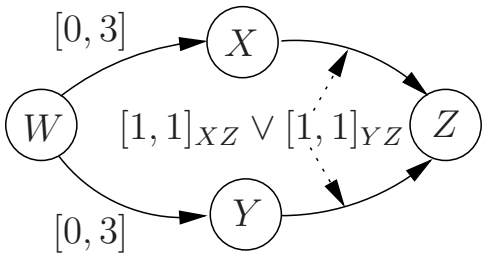


Figure 4: Example DTN.

2.1.3 DTN

The final extension to the constraint definition is to allow arbitrary disjunctions of simple temporal constraints (not necessarily over the same two events) [Stergiou and Koubarakis, 2000]. When we relax the constraints to allow arbitrary disjunctions, we have a *Disjunctive Temporal Network* (DTN).[1] With the increased flexibility, we have the relation that an STN is a special case of a TCSN, which is in turn a special case of a DTN. Formally, the constraints in a DTN are defined as follows:

Definition 3 (Disjunctive Temporal Constraint)**.** For events $X_1, \cdots X_n$ and $Y_1, \cdots Y_n$ (allowing multiple events to be equal), $\{[l_1, u_1]_{X_1Y_1}, \cdots, [l_n, u_n]_{X_nY_n}\}$ is the disjunctive temporal constraint that restricts the timing of the events to be,

$$l_1 \leq T_{Y_1} - T_{X_1} \leq u_1 \vee \cdots \vee l_n \leq T_{Y_n} - T_{X_n} \leq u_n$$

Intuitively, a disjunctive temporal constraint in a DTN allows for the disjunction of any number of arbitrary simple temporal constraints – satisfying just one of them leads to the disjunctive temporal constraint being satisfied.

Example 3 (DTN)**.** Figure 4 shows a simple DTN example with 4 events (W, X, Y, Z), and 3 temporal constraints:

- $\{[0, 5]_{WX}\}$
- $\{[0, 5]_{WY}\}$
- $\{[1, 1]_{XZ}, [1, 1]_{YZ}\}$

Here, the disjunctive constraint ensures that either event X or Y occurs precisely 1 time unit before Z. Determining if a DTN is consistent is referred to as the *Disjunctive Temporal Problem* (DTP), and our example DTN is consistent. A confirmation of consistency is the setting of the time points: $T_W = 0$, $T_X = 2$, $T_Y = 3$, and $T_Z = 4$.

For both TCSNs and DTNs, selecting a single disjunct from every disjunctive constraint results in a set of simple temporal constraints, which in turn can be viewed as an STN. For every such selection, we refer to the resulting STN as the *component STN* and the problem of determining if the component STN is consistent as the *component STP*. This concept will be central to the approaches for solving TCSPs and DTPs.

[1] Note that any arbitrary well-formed formula over simple temporal constraints can be compiled down to conjunctive normal form, resulting in a DTN. [Stergiou and Koubarakis, 2000]

2.2 Uncertainty

The original motivation for delaying the selection of event timing was to be robust during online execution. Things may change at run-time, which will require events to be executed earlier or later. Delaying the selection of event execution time for as long possible helps mitigate the problem of replanning from scratch. To expand on this notion of uncontrollable behaviour, [Vidal and Fargier, 1999] introduced the concept of a *contingent constraint*. Similar to a simple temporal constraint, a contingent constraint restricts the the relative time between two events. However, rather than allowing the dispatcher to choose the length of this edge, nature is assumed to select the duration.

Each of the previously introduced formalisms were extended to include contingent constraints. The problem of network consistency, however, becomes less relevant. A consistent network may not work in practice if nature chooses some duration that violates the solution that was found to be consistent. For this reason, the notion of *controllability* was introduced. Three different levels of controllability, described further in Section 2.3, encompass the various conditions in which we may want to dispatch the plan to ensure that a full execution will always exist (regardless of the durations of the contingent constraints).

2.2.1 STNU

Extending an STN to include a model of uncertainty involves partitioning both the set of events and the set of temporal constraints to handle those events and durations that are controlled by nature and not the agent. The definitions and notation found in this section are a selection from the literature primarily from [Vidal and Fargier, 1999] and [Hunsberger, 2009]. We have altered some of the notation to unify the work with the other sections, and to have a common way of describing the various approaches that we will see in Section 3. Like an STN, an STNU has simple temporal constraints between actions, but also has *contingent constraints* whose durations are defined by nature. Formally, we define a contingent constraint as follows:

Definition 4 (Contingent Temporal Constraint)**.** For events X and Y, $[[l, u]]_{XY}$ is the contingent temporal constraint that restricts the timing of events X and Y to be,

$$l \leq T_Y - T_X \leq u$$

The values of l and u may be any real number such that $0 < l < u < \infty$.

A contingent temporal constraint represents an uncontrollable duration. We use ω_{XY} to denote the observed length of the contingent constraint $[[l, u]]_{XY}$ (i.e., $\omega_{XY} = T_Y - T_X$). Note that the only information available to the compiler/dispatcher (until Y occurs) is that $\omega_{XY} \in [[l, u]]_{XY}$.

The set of events is also partitioned into *executable* and *observed* events. However, the distinction between the two depend on the type of temporal constraints they belong to. Formally, a Simple Temporal Network with Uncertainty (STNU) is as follows:

Definition 5 (STNU)**.** A *Simple Temporal Network with Uncertainty* (STNU) is a 4-tuple, $\mathcal{N} = (\mathcal{X}_e, \mathcal{X}_o, C_f, C_c)$:

- $\mathcal{X}_e$: Set of *executable events*.
- $\mathcal{X}_o$: Set of *observed events*.
- C_f: Set of *simple temporal constraints*.
- C_c: Set of *contingent temporal constraints*.

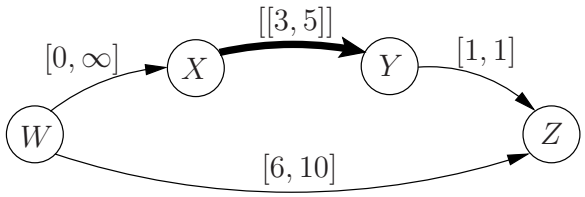


Figure 5: Example STNU.

The set $\mathcal{X}_o$ is uniquely defined to be the endpoints of the constraints in C_c, and we forbid any observed event to be involved in more than one contingent constraint.

We refer to the constraints in C_f as *free constraints* (to distinguish them from contingent ones). We restrict the observed events from taking part in multiple contingent constraints to both simplify the exposition, and prevent two contingent constraints from leading to the same event (a situation where nature may cause an inconsistency). If we need to sequence multiple contingent constraints consecutively, then we simply add the free constraint $[0, 0]$ between the end of one contingent constraint and the start of the next.

Example 4 (STNU)**.** Figure 5 shows a simple example of an STNU that includes a contingent constraint between the events X and Y. This constraint may represent the uncontrollable duration of an action that starts with X and ends with Y. Note here that we do not distinguish between the events $\mathcal{X}_e$ and $\mathcal{X}_o$ since the distinction is implicit with the use of the contingent constraints.

Note that we allow free constraints between events of any type. These simply restrict the relative timing of the events, and we must handle the uncertainty of observed events in any approach for reasoning with an STNU. To further describe the methods that handle uncertainty, we introduce additional terminology surrounding the scheduling of events in an STNU:

Definition 6 (Complete / partial control sequence)**.** A *control sequence* is a set of assignments to executable events. A control sequence is said to be *complete* if it is an assignment to every executable event in $\mathcal{X}_e$. Otherwise, the control sequence is *partial*.

Definition 7 (Complete / partial situations)**.** Given $C_c = \{[[l, u]]_{X_1Y_1}, \cdots, [[l, u]]_{X_nY_n}\}$, we define the *space of complete situations* of the STNU to be:

$$\Omega = [[l, u]]_{X_1Y_1} \times \cdots \times [[l, u]]_{X_nY_n}$$

The set $\omega = \{\omega_{X_1Y_1} \in [[l, u]]_{X_1Y_1}, \cdots, \omega_{X_nY_n} \in [[l, u]]_{X_nY_n}\} \in \Omega$ is called a *complete situation* of the STNU. Any subset of a complete situation is a partial situation.

Intuitively, a complete situation (also referred to as simply a *situation*) represents one potential configuration of the entire set of contingent constraints. For any situation ω, the STNU reduces to a classic STN, $\mathcal{N}_\omega$. A solution to the corresponding STP is a complete control sequence that satisfies the free temporal constraints in $\mathcal{N}_\omega$.

Definition 8 (Projection)**.** For every $\omega \in \Omega$, $\mathcal{N}_\omega$ is referred to as the *projection* of the STNU $\mathcal{N}$ in the situation ω, and we construct it by replacing every contingent constraint $[[l, u]]_{XY}$ with the free constraint $[\omega_{XY}, \omega_{XY}]$.

Note that while the projection of a complete situation is a classical STN, the projection of a partial situation remains an STNU. Like its controllable counterpart, there is a corresponding *Simple Temporal Problem with Uncertainty* (STPU), but as mentioned previously the STPU involves controllability rather than consistency, and will be covered in Section 2.3.

2.2.2 TCSNU

While it is natural to extend TCSNs to include uncertainty (i.e., add the possibility of a disjunct being a contingent constraint), research in the field has focused primarily on the more general case of arbitrary disjunctions with uncertainty.[2] Recall that only considering DTNUs is not a restriction, as a temporal constraint satisfaction network (with uncertainty) is just a special case of a disjunctive temporal network (with uncertainty). The only difference is that constraints found in a disjunction must be over the same two events. Like STNUs, the TCSN with uncertainty (TCSNU) has a problem counterpart; the *Temporal Constraint Satisfaction Problem with Uncertainty* (TCSPU).

2.2.3 DTNU

DTNs have been extended to handle uncertainty analogous to STNUs [Venable and Yorke-Smith, 2005], giving us a *Disjunctive Temporal Network with Uncertainty* (DTNU). Rather than having only disjunctions of simple temporal constraints, we introduce contingent constraints. We adopt the similar notation from STNUs ($\mathcal{X}_e, \mathcal{X}_o$, contingent and simple temporal constraints), but to adequately classify the types of constraints that are allowed, we first define three notions of constraints found in an STNU:

1. *Contingent STNU constraints*: These are the contingent temporal constraints as defined previously.
2. *Executable STNU constraints*: These are the constraints between an event in $\mathcal{X}_o$ and an event in $\mathcal{X}_e$.
3. *Executable STN constraints*: These are the constraints between two events in $\mathcal{X}_e$.

With these notions in hand, we proceed to describe the types of disjunctive constraints one may find in a DTNU (besides disjunctions with only one disjunct). Early work simply categorized the three obvious types of disjunctions: *executable* (simple temporal constraints only), *contingent* (contingent temporal constraints only), and *semi-executable* (a mix of both simple and contingent constraints). Later, however, the notions were revised in [Peintner et al., 2007] to restrict the types of disjunctions further, and clarify their meaning. The four types proposed are as follows:

1. *DTN*: A disjunction of two or more STN constraints.
2. *Executable DTNU*: A disjunction of two or more executable STNU constraints.
3. *Mixed executable DTNU*: A disjunction of two or more executable STN and STNU constraints.
4. *Contingent DTNU*: A disjunction of two or more contingent STNU constraints.

Note the absence of a mixed contingent constraint. The argument for leaving this type out is that the philosophy behind use of a contingent constraint is to model uncertainty in the world that we know will occur. If the agent can choose to satisfy some other disjunct in mixed contingent constraint (i.e., one that is

[2] One exception being [Venable et al., 2010], where the problem they consider dynamic controllability of a TCSNU.

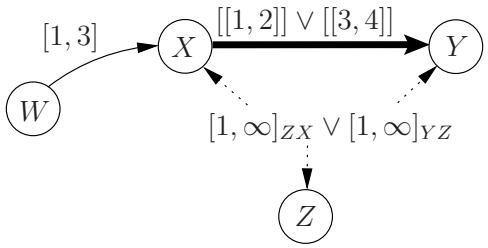


Figure 6: Example DTNU.

a simple temporal constraint), then it goes against the intuition of a contingent constraint's purpose. There is a further assumption placed on the Contingent DTNU constraints: the disjuncts must all be uniform in the events they talk about (similar to a TCSN constraint). This simplification was imposed for the intuition that uncertainty added to the DTN is only in the form of the duration of some process (i.e., the relative time between two events). As such, in a DTNU, contingent constraints will only appear in a disjunction when the other disjuncts are contingent constraints for the same pair of events.

Example 5. Figure 6 shows a simple example of a DTNU. The constraint between events X and Y is a contingent DTNU constraint that can be interpreted as a process that takes either 1-2 time units or 3-4 time units, and nature will decide which disjunct (and duration) is chosen. The constraint between X, Y, and Z is an example of a mixed executable DTNU constraint, and can be interpreted as event Z must not be executed within 1 time unit of the XY process (i.e., it should happen at least 1 time unit before X or after Y).

Similar to STNUs and TCSNUs, the corresponding controllability problem of a DTNU is referred to as the *Disjunctive Temporal Problem with Uncertainty* (DTPU).

2.3 Controllability

When the duration of a temporal link is determined by nature, the notion of consistency makes little sense – it would only indicate that there is a possibility nature will allow the network to be consistent. We instead use a notion of *controllability*. Depending on when we may know about the choices of nature (i.e., the lengths of the contingent links in the network), we have three notions of controllability: *strong*, *weak*, and *dynamic*.

In this section, we give the high level description for the three forms of controllability. All three apply to formalisms that include uncertainty; namely STNUs, TCSPUs, and DTNUs. For a temporal network $\mathcal{N}$, we have the following relation between the three forms of controllability:

$$\mathcal{N} \text{ is } \textit{strongly controllable} \Rightarrow \mathcal{N} \text{ is } \textit{dynamically controllable} \Rightarrow \mathcal{N} \text{ is } \textit{weakly controllable}$$

2.3.1 Strong Controllability

The most restrictive notion of controllability is *strong controllability*. Intuitively, it is similar to computing a conformant plan (in this case, a conformant schedule) [Russell and Norvig, 2003]. Formally, we define strong controllability as follows:

Definition 9 (Strong controllability [Vidal and Fargier, 1999])**.** A network with uncertainty (i.e., contains at least one contingent constraint) is *strongly controllable* iff there is a setting to the executable time points such that every possible projection is consistent.

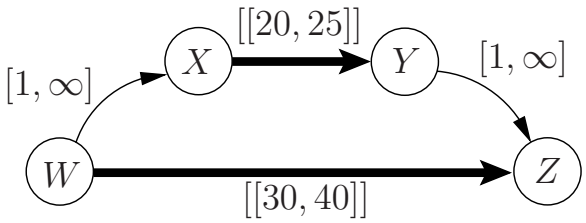


Figure 7: Example of an STNU that is strongly controllable.

Recall that the projection of an STNU (resp. DTNU) is the STN (resp. DTN) that results from setting the duration of every observed event. For a DTNU, a projection is defined analogously to a STNU, and strong controllability holds when there is a setting of the executable time points such that in every projection at least one disjunct from every disjunction is true.

Note that we must have a consistent network with a choice of time points, regardless of what nature decides. While this notion of controllability is usually too restrictive, it does have application in situations where we want to guarantee that a plan will be executed properly, and the agent cannot modify its plan in an online manner (e.g., durations cannot be observed at all).

For illustrative purposes, we show an example STNU in Figure 7 that is strongly controllable. Note that the executable time points are W and X. Assigning $T_W = 0$ and $T_X = 1$ satisfies the constraint between W and X. Regardless of the durations selected for edges WZ and XY, we can see that $T_Y \leq 26$ and $T_Z \geq 30$. Thus, we know that the constraint between Y and Z will be satisfied, and the network consistent.

2.3.2 Weak Controllability

To relax the restrictive nature of strong controllability, we introduce a notion that is more situation dependent: *weak controllability*. Intuitively, weak controllability simply means that there is always a way to make the network consistent if we know in advance what nature will decide. Formally, it is defined as follows:

Definition 10 (Weak controllability [Vidal and Fargier, 1999])**.** A network with uncertainty is *weakly controllable* iff for every possible projection, there exists a setting to the executable time points that makes the network consistent.

In the case of weak controllability, the agent may choose the timing for every (executable) time point *after* nature's decisions have been set. This property may be relevant when the entire set of observed durations is given before execution begins (possibly *just* before the execution begins).

In Figure 8 we show a simple example of an STNU that is weakly controllable but not strongly controllable. Here, we want the process XY to complete exactly one time unit after the process WZ finishes. There is no setting of executable events X and W that will ensure we can do so for every choice of duration, but if we know the durations in advance then we can easily choose the appropriate times.

2.3.3 Dynamic Controllability

Arguably, the most natural way to consider the notion of controllability is to allow the agent to make decisions in an online fashion; using only the information about nature's decisions that it has seen already. This observation is the motivation for *dynamic controllability*. Since we are now concerned with intermediate points in time for the decision making of the agent, we must introduce additional notation to handle the

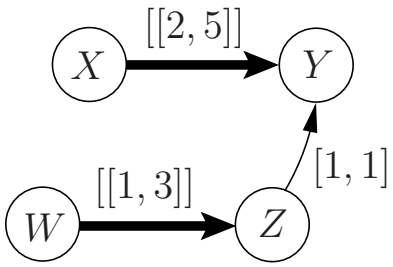


Figure 8: Example of an STNU that is weakly controllable.

formalization of dynamic controllability. The majority of the literature build on the definitions presented in [Vidal and Fargier, 1999], but as [Hunsberger, 2009] points out, there is a fundamental flaw with the definition of dynamic controllability.[3] As such, we will use the formalism found in [Hunsberger, 2009]. We present the formalism for STNUs, but in general the same principles apply when extending it to TCSPUs and DTPUs.

Recall that a *schedule* is a complete setting of the times for every event in a network. We introduce a notion to refer to the contingent link durations that have occurred in the past:

Definition 11 (Pre-history)**.** Given an STNU, $\mathcal{N} = (\mathcal{X}_e, \mathcal{X}_o, C_s, C_c)$, a schedule T, and some time value $k \in \mathbb{R}$, then $T^{<k}$ is the *pre-history of T relative to k*. The pre-history gives the durations of the contingent links in $\mathcal{N}$ that finish before k in T:

$$T^{<k} = \{(XY, \omega_{XY}) \mid XY \in C_c, T_Y < k\}$$

Definition 12 (Dynamic Execution Strategy (DES))**.** For an STNU, $\mathcal{N} = (\mathcal{X}_e, \mathcal{X}_o, C_s, C_c)$, an *execution strategy* is a mapping $S : \Omega \rightarrow \mathcal{T}$, from situations to schedules. The schedule is *viable* iff $\forall \omega \in \Omega$, the schedule $S(\omega)$ is consistent with the projection $\mathcal{N}_\omega$. S is said to be a *dynamic execution strategy* (DES) for $\mathcal{N}$ if for any situations $\omega', \omega'' \in \Omega$, and executable time point $X \in \mathcal{X}_e$,

$$[S(\omega')]_X = k \ \ and \ \ [S(\omega')]^{<k} = [S(\omega'')]^{<k} \Longrightarrow [S(\omega'')]_X = k$$

Intuitively, if the strategy S in situation ω' assigns the time of k to the event X, then the strategy must also assign k to X in any situation that shares the same pre-history (relative to k). We can now define dynamic controllability formally as follows:

Definition 13 (Dynamic controllability)**.** An STNU $\mathcal{N}$ is *dynamically controllable* iff there exists a viable dynamic execution strategy for $\mathcal{N}$.

The use of a dynamic execution strategy captures the essence of what we want from an agent executing the plan – to dynamically choose the most appropriate action based on the durations of previous contingent links, and guarantee that we can continue executing regardless of the timing of future observed events. In Section 3.4 we shall see that dynamic controllability is usually computed by verifying a necessary and sufficient condition on the network. Subsequent dispatching algorithms are then described to handle the execution of the events in a safe manner.

Note that the previous example (Figure 8) is *not* dynamically controllable: there is no way to even decide if X or W should be executed first without knowing how long each of the contingent links will be. In Figure 9 we give an example of a network that is dynamically controllable, but not strongly controllable.

[3] Although a flaw was discovered, the results that used the previous definition continue to hold under the new formalism.

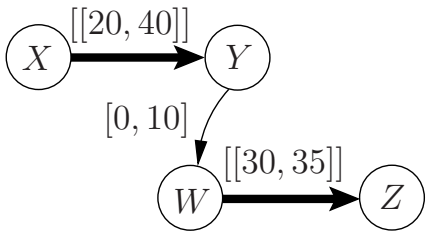


Figure 9: Example of an STNU that is dynamically controllable.

Consider the duration XY to be the time it takes to prepare a meal, and WZ to be the time it takes to consume it. Ideally, we want to eat while the meal is still warm (hence the $[0, 10]_{YW}$ constraint). Note, however, that there is no way to know for sure when we should start eating (i.e., T_W) before knowing how long it will take to prepare the meal. Thus, this network is not strongly controllable. On the other hand, once we know the length of the dinner preparation, we can then decide on when the meal should be consumed.

Notice that the decision for T_W really only depends on the observed events that occurred in the past (i.e., the time it took to prepare the meal). This situation exemplifies the nature of dynamic controllability.

3 Approaches to Compiling and Dispatching

In this section, we describe the methods for checking consistency and controllability properties of the various formalisms. Both verifying the property and dispatching the network are considered.

3.1 Consistency

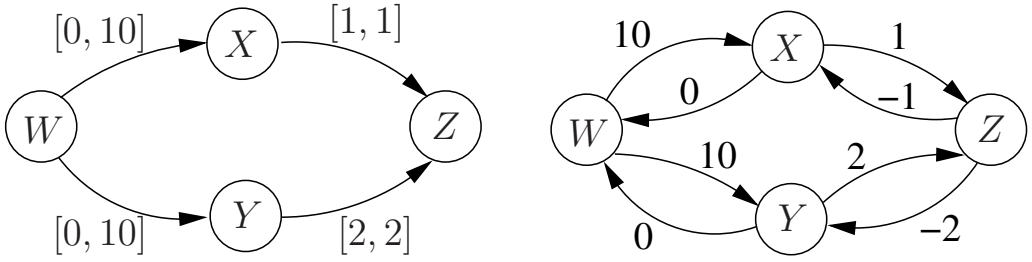
As we saw in Section 2.1, for networks without uncertainty we are interested in knowing if the network is consistent (i.e., can it be effectively dispatched). In this section, we consider the problem of determining consistency for the STP, DTP, and TCSP formalisms.

3.1.1 STP

The problem of checking for consistency in an STP is one of the most prevalent among the approaches in this section, due largely in part to the reuse of STP consistency as a subroutine in so many applications; from maintaining consistent plans during plan generation [Younes and Simmons, 2003] to effective techniques for testing controllability [Hunsberger, 2010]. All of the approaches to checking STP consistency start from the same canonical form: the STP is converted into the *distance graph*, where nodes correspond to the events in the STP and edges correspond to upper bounds on the time between one node and another (this splitting of constraints was introduced in Section 2.1.1). Figure 10a shows an example STN we have already seen, while Figure 10b shows the corresponding distance graph.

All of the approaches involve modifying the distance graph into what is known as *dispatchable form*. A distance graph is dispatchable if it can be effectively executed by Algorithm 1. Note that the complexity of propagation in the algorithm depends on the number of neighbours an event has.

In Algorithm 1, A contains the activated events (those that are eligible to execute when the time fits its current bounds), S contains those events that have already been executed, and $l(X)$ ($u(X)$) gives the lower (upper) bounds on the time event X is executed. Lines 5-7 pick an eligible event arbitrarily and executes it,



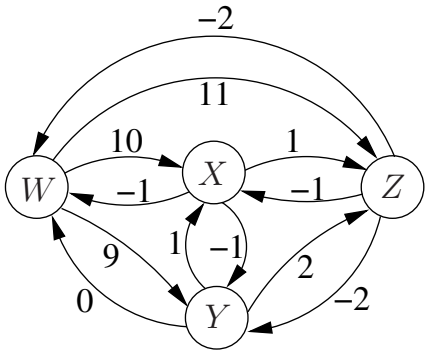
(a) Example STN. (b) Example of a distance graph for an STN.

Figure 10

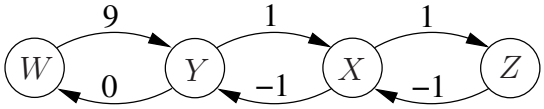
Algorithm 1: STN Dispatching Algorithm

Input: Distance graph, $\langle V, E, w\rangle$, where V is the set of events, E is the set of edges, and $w : E \to \mathbb{R}$ is a weighting of the edges such that $(X, Y) \in E$ represents the constraint $Y - X \leq w(X, Y)$

$A = \{\text{start\_time\_point}\}, \quad t = 0, \quad \text{and } S = \emptyset$
$l(\text{start\_time\_point}) = u(\text{start\_time\_point}) = 0$
$\forall x \in V \setminus \{\text{start\_time\_point}\}, l(x) = 0 \text{ and } u(x) = \infty$
while $S \neq V$ **do**
  Pick some $X \in A$ such that $l(X) \leq t \leq u(X)$
  $T_X = t$
  $S = S \cup \{X\}$
  foreach $(X, Y) \in E$ *and* $Y \notin S$ **do**
    $u(Y) = \min\{u(Y), t + w(X, Y)\}$
  foreach $(Y, X) \in E$ *and* $Y \notin S$ **do**
    $l(Y) = \max\{t, l(Y), t - w(Y, X)\}$
  $A = \{W \in V - S \mid \forall (W, Z) \in E, [w(W, Z) < 0 \Rightarrow Z \in S]\}$
  Wait until $\min_{X \in A} l(X) \leq t \leq \min_{Y \in A} u(Y)$



(a) Example of the all-pairs-shortest-path for a simple STN.



(b) Example of the d-graph for a simple STN.

Figure 11

while lines 8-11 propagate the bounds information to the direct neighbours of the event being executed. Line 12 sets A to be all those events that can now be executed (all events necessarily preceding them have been executed). Finally, line 13 waits an undefined length of time until we can at least execute some event, and no later than the minimum deadline of another event. This pause of arbitrary length gives the agent flexibility at execution time to delay events to handle external contingencies. These are unplanned contingencies left out of the model however (as supposed to the modelled contingencies of STNUs).

As noted above, not every distance graph can be dispatched with the Algorithm. The goal of compilation then, is to transform a distance graph into one that *can* be dispatched. Arguably one of the simplest approaches in the literature, described in [Muscettola et al., 1998], involves running an all-pairs-shortest-path algorithm on the distance graph. This process generates what is known as the *d-graph*, and does so in $O(n^3)$ time (where n is the number of events in the STN). It was shown that the network is consistent iff there is no negative cycle in the d-graph. It was further shown that the d-graph of a consistent STN is guaranteed to be dispatchable.

[Muscettola et al., 1998] went on to show how the graph can be simplified by removing redundant edges in the dispatchable d-graph. Motivated by the complexity of the above algorithm (propagations going to every neighbour of an executed event), they propose a set of dominance relations that allows one to soundly remove edges from the d-graph (noting that a consistent d-graph satisfies the triangle inequality):

1. A non-negative edge AC is upper dominated by another non-negative edge BC iff $w(A, B) + w(B, C) = w(A, C)$
2. A negative edge AC is lower-dominated by another negative edge AB iff $w(A, B) + w(B, C) = w(A, C)$

Any edge that is dominated (either upper or lower), is redundant and can be safely removed from the d-graph. Reducing the number of edges in the d-graph allows for quicker dispatch, as the loops on lines 8 and 10 in Algorithm 1 are reduced. To illustrate the potential of reductions, we first show the all-pairs-shortest-path graph in Figure 11a. After removing the redundant edges, we are left with the d-graph shown in Figure 11b.

To improve on the $O(n^3)$ complexity of the all-pairs-shortest-path algorithm used to determine STN consistency, [Xu and Choueiry, 2003] introduced a method that extends the STN to be chordal, and determined consistency of this chordal graph in time quadratic in the number of triangles in the graph. Later,

[Planken et al., 2008] improved on this method by taking advantage of a graph-theoretic process called *simplical elimination*. Their technique reduces the complexity to be linear in the number of triangles, and the overall complexity ends up being $O(n \cdot (w^*(d))^2)$ (where $w^*(d)$ is a measure of width in the graph and depends on an ordering of the vertices d). The complexity may still be $O(n^3)$ in the worst case (when the STN is a complete graph), but in general $w^*(d) \ll n$. Here, we present the most recent approach by Planken et al, and begin by introducing the graph theoretic concepts required.

Definition 14 (Chordal Graph)**.** Let $G = \langle V, E \rangle$ be a graph, and $(v_1, v_2, \cdots, v_k, v_1)$ a cycle of k vertices in the graph with $k > 3$. A *chord* is any edge between v_i and v_j such that they are non-adjacent in the cycle, and $1 < j - i < k - 1$. A *chordal graph* is a graph where every cycle of length greater than 3 has a chord.

In general, a chordal graph will have far fewer edges than a complete graph. We can add edges to a non-chordal graph to make it chordal (referred to as *filling*), and if the original graph is sparse then it usually is the case that the corresponding chordal graph will be sparse as well.

Definition 15 (Simplical Elimination)**.** In a graph $G = \langle V, E \rangle$, a vertex $v \in V$ is *simplical* if the set of its neighbours induces a clique (i.e., $\forall x, y, \in V, (v, x) \in E \wedge (v, y) \in E \Rightarrow (x, y) \in E$). Let $d = (v_n, \cdots, v_1)$ be a complete ordering of V, and G_i denote the subgraph induced by the vertices $V_i = \{v_1, \cdots, v_i\}$. The ordering d is a *simplical elimination ordering* of G if every vertex v_i is a simplical vertex of G_i.

From graph theory, we know that a graph is chordal if and only if it has a simplical elimination ordering. This property allows us to construct a chordal graph by heuristically finding a simplical elimination ordering, adding edges when no simplical vertex exists to remove. Once a simplical elimination ordering is computed, and the graph is in chordal form, we can do a two-pass algorithm to ensure that the network is consistent, and place it in a dispatchable form for Algorithm 1. The fill edges that are added will start with open bounds ($[-\infty, \infty]$) and will be strengthened by the consistency check procedure. We present this two-pass algorithm, P^{3}C, in Algorithm 2.

Algorithm 2: P^{3}C Algorithm for STN Consistency

Input: A chordal STN $S = \langle V, E \rangle$ with a simplical elimination ordering $d = (v_n, \cdots v_1)$
Output: A dispatchable STN, or INCONSISTENT

```
1 for k = n ··· 1 do
2     foreach i, j < k s.t., (v_i, v_k) ∈ E ∧ (v_k, v_j) ∈ E do
3         w(v_i, v_j) = min{w(v_i, v_j), w(v_i, v_k) + w(v_k, v_j)}
4         if w(v_i, v_j) + w(v_j, v_i) < 0 then
5             return INCONSISTENT
6 for k = 1 ··· n do
7     foreach i, j < k s.t., (v_k, v_i) ∈ E ∧ (v_i, v_j) ∈ E ∧ (v_j, v_k) ∈ E do
8         w(v_i, v_k) = min{w(v_i, v_k), w(v_i, v_j) + w(v_j, v_k)}
9         w(v_k, v_j) = min{w(v_k, v_j), w(v_k, v_i) + w(v_i, v_j)}
```

Lines 1-5 enforce what is called *direct path consistency*, and was originally proposed in [Dechter et al., 1989]. It is sufficient to determine if the STN is inconsistent, but further enables the rest of the algorithm to work correctly when the ordering of vertices is a simplical elimination ordering. Lines 6-9 enforce what is called *partial path consistency*, and results in the STN being placed in dispatchable form.

As was mentioned previously, the complexity of this approach can (in the worst case) be $O(n^3)$. However, it is usually much smaller since Algorithm 2 depends on a notion of graph width. Specifically, the complexity of Algorithm 2 is $O(n \cdot (w^*(d))^2)$ where $d = (v_n, \cdots, v_1)$ and $w^*(d)$ is defined as:

$$w^*(d) = \max_i |\{(v_i, v_j) \in E \mid j < i\}|$$

3.1.2 DTP

In Section 2.1.3 we saw that a DTN is consistent if and only if one of its component STPs is consistent (recall that a component STP is a selection of a disjunct from every disjunctive constraint in the DTN). This observation is what motivates the naive approach for determining DTN consistency presented in [Tsamardinos et al., 2001]. Intuitively, the approach maintains information for every component STP that remains valid while dispatching, and ensures that any action selection leaves at least one component STP that remains feasible.

The approach of Tsamardinos et al. is to enumerate all of the component STPs, compute the dispatchable form of each, and use proper data structures to maintain the information needed for online dispatching. The two key data structures that are used for dispatching are the *execution table* and *deadline formula*. Before describing these, we introduce some key notation.

An STP event is enabled iff all of the events that are constrained to occur before it have already been executed. An event in a DTN is *enabled* iff it has a consistent component STP in which the event is enabled. The *DTN time window* of an event, is the union of the time windows over every component STP. We now define the main data-structures that are used:

Definition 16 (Execution Table)**.** The *execution table* (ET) provides information about when an action is enabled (i.e., can be executed). It consists of a list of ordered pairs, one for each enabled event. The first entry in the pair specifies the event, and the second is a set of time intervals that correspond to the event's time window.

Definition 17 (Deadline Formula)**.** In order to execute the DTN properly, we keep track of the minimal conditions of events that must occur before a certain time, referred to as the *deadline formula* (DF). The DF consists of a positive formula in conjunctive normal form over events in the DTN, and a single point of time. The interpretation is that the formula must hold (one event from every disjunction must be executed) prior to the specified time.

The ET is populated initially from the component STP solutions, and DF is computed by enumerating the minimum set covers over upper bound deadlines that span the component STPs.[4] As the dispatcher executes events, the component STPs are enumerated, and time bounds in ET are updated. The DF is also updated to reflect the execution of new events (which may satisfy certain conjuncts in the formula). To illustrate the data that may be found in ET and DF, consider the example shown in Figure 12. In this simple example, there are four disjunctive constraints and sixteen possible component STPs, but only four of them are consistent (shown in Figure 13).

From the consistent STPs, and considering event W to be the starting point at $t = 0$, we have the following tuples in ET for the time windows of events X, Y, and Z:

- $\langle X, \{[5, 10], [15, 20]\}\rangle$

[4]Full details of this procedure is out of the scope of this review.

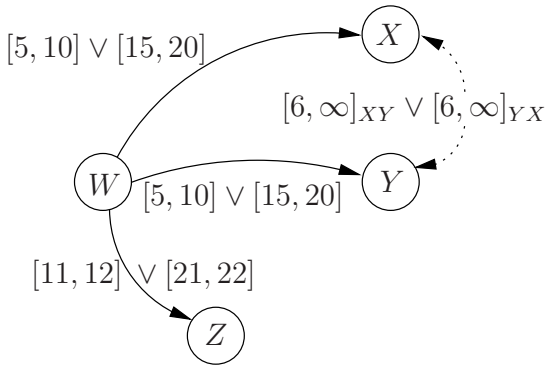


Figure 12: Example DTN.

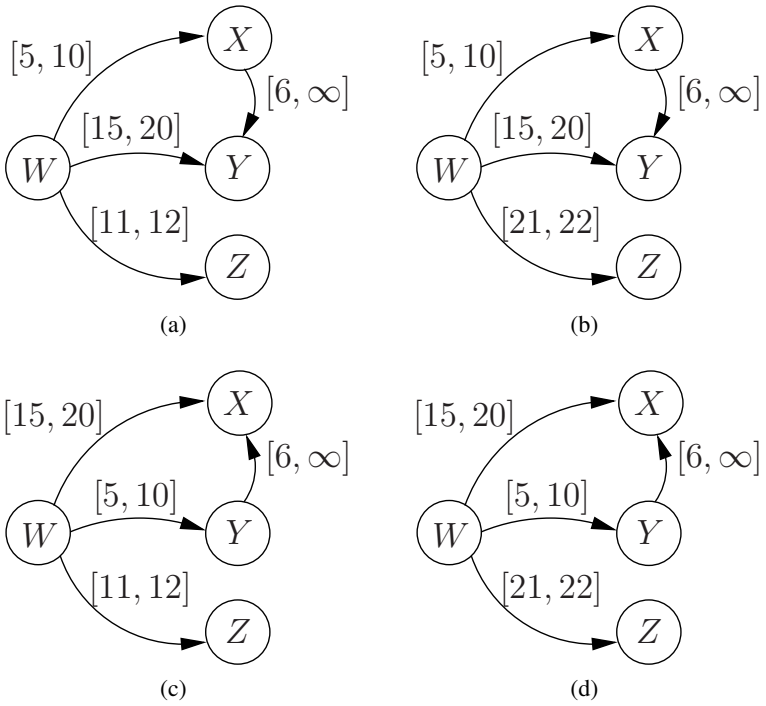


Figure 13: Four consistent component STPs of the DTN in Figure 12.

- $\langle Y, \{[5, 10], [15, 20]\}\rangle$
- $\langle Z, \{[11, 12], [21, 22]\}\rangle$

The DF for this example would be initialized to $\langle P \vee Q, 10\rangle$. The DF ensures that we execute either X or Y before we pass time 10 (otherwise, both would be required to be within the interval of $[15, 20]$ which causes a contradiction).

The drawback of this approach is the overhead required beforehand, and during execution, to enumerate every component STP (as there may be exponentially many of them). To overcome this problem, [Conrad et al., 2009] introduced a technique that leverages *Assumption Based Truth Maintenance Systems* (ATMS) (further described in [Conrad and Williams, 2011]). The general idea is to represent all of the component STPs implicitly through careful use of a labelling protocol inspired by ATMS research. Compilation and dispatching are conducted on a labelled variation of an STN, and the labels are updated during execution to keep the network consistent. The key structure used for compiling and dispatching is a *labelled distance graph*, and is defined as follows:

Definition 18 (Labelled Distance Graph)**.** Each disjunctive constraint in the DTN is associated with a finite domain variable x_i, and it its value corresponds to which disjunct is satisfied. We refer to the x_i's as *choice variables*, and X is the set of all choice variables.[5] A set of assignments to some subset of the choice variables is referred to as an *environment*. In this context, we define a *labelled STN* as a regular STN, but each constraint is "labelled" with an environment (possibly the empty one, $\{\}$), and there may be multiple constraints between two events. Finally, we define a *labelled distance graph* as a triple, $\langle V, W, X\rangle$, where V is the set of events, X is the set of choice variables in our DTN, and W maps pairs of vertices and environments to a weight (i.e., $V \times V \times \mathcal{E} \rightarrow \mathbb{R} \cup \infty$, where $\mathcal{E}$ is the set of all possible environments). Where convenient, we will also say that $\langle a, e\rangle \in W(A, B)$ if $W(A, B, e) = a$.

Similar to the unlabelled version, the weights on a labelled distance graph represent the inequalities. The only difference is that they are now predicated on choice variables (i.e., selection of a disjunct) and there may be multiple labels between two events. We could view the labelled distance graph as a multi-graph, but for clarity reasons it is better to consider it as a directed graph with an edge between X and Y iff there is some environment $e \in \mathcal{E}$ such that $W(X, Y, e) \neq \infty$. To operate with environments, we define some basic concepts inherited from the ATMS literature:

Definition 19 (Subsumption and Union)**.** We say that an environment e *subsumes* another environment e', denoted as $e \prec e'$, if for every assignment $x_i = d_i \in e$, the same assignment exists in e' (i.e., $x_i = d_i \in e'$). For example, the environment $\{x_1 = 2, x_3 = 1\}$ subsumes the environment $\{x_1 = 2, x_2 = 1, x_3 = 1\}$. The *union* of two environments, denoted $e \cup e'$, is the union of all assignments found in both environments. If there are contradictory assignments to a variable (e.g., $x_1 = 2 \in e, x_1 = 3 \in e'$), then there is no valid union, and $\perp$ is returned signifying that there is no environment in which both e and e' can hold simultaneously.

Environments are not only used for labelling edges in the labelled distance graph: similar to environments in an ATMS system, an environment can represent a situation that should be avoided. A *conflict* is an environment that entails an inconsistency of some sort. In this context, it will be a set of choices to the disjunctive constraints that lead to an inconsistent component STP. Note that a conflict may represent an exponential number of component STPs, all of which are inconsistent. To adequately reason about sets of environments, and the values they entail, we introduce the notion of a labelled value set:

[5] Note that a setting to every choice variable corresponds to a component STP.

Definition 20 (Labelled Value Set)**.** We first define a *domination function* as a binary function $f(a, a')$ that induces a complete ordering over the domain of its elements ($f(a, a)$ is assumed to be false). An example for the domain of real numbers is $f(a, a') = a < a'$. A *labelled value set* for a domination function f is a set A of pairs $\langle a, e\rangle$ where a is in the domain of f, $e \in \mathcal{E}$, and $\forall\langle a, e\rangle, \langle a', e'\rangle \in A, \neg(f(a, a') \wedge e \prec e')$.

Intuitively, a labelled value set for a domination function f contains tuples of value-environment pairs such that no pair "dominates" another – i.e., the value dominates and the environment subsumes. This minimal form allows us to maintain only the strongest assumptions, and discard the rest. When new pairs are added to a labelled value set, dominated pairs are automatically removed. We further define a query operation for the set

Definition 21 (Labelled Value Set Query)**.** We define $A(e)$ to be the *labelled value set query* that maps an environment to the dominant value from the pairs in A such that the pair's environment subsumes e. Formally, $A(e)$ is defined as:

$$A(e) = a^* \text{ s.t. } \langle a^*, e^*\rangle \in A \wedge e^* \prec e \wedge \forall\langle a_i, e_i\rangle \in A, e_i \prec e \rightarrow [f(a^*, a_i) \vee a^* = a_i]$$

Note that e_i need not subsume e^*. For example, consider the following scenario:

- $A = \{\langle 3, \{x_1 = 2\}\rangle, \langle 4, \{x_2 = 3\}\rangle, \langle 2, \{x_1 = 3\}\rangle\}$
- $f(a, a') = a < a'$
- We want to query $A(\{x_1 = 2, x_2 = 3\})$

Here, the environment in the first two pairs subsume the query, but neither subsumes the other. Further, the environment in the third pair does not subsume the query, and so we only consider the first two pairs. Given that the function is $<$, we have $A(\{x_1 = 2, x_2 = 3\}) = 3$.

Now that we have formalized labelled value sets, we can define how operations between them will work.

Definition 22 (Labelled Value Set Operations)**.** For two labelled value set pairs $\langle a, e\rangle$ and $\langle a', e'\rangle$, and any deterministic function g over two values, the *value set pair operation* is defined to be $\langle g(a, a'), e \cup e'\rangle$. For two value sets A and A', and a binary deterministic function g, we denote the *value set operation* $g(A, A')$ to be:

$$\{\langle g(a, a'), e \cup e'\rangle \mid \langle a, e\rangle \in A, \langle a', e'\rangle \in A'\}$$

As with any other labelled value set, $B = g(A, A')$ is presumed to be minimal (no pair dominates another). To illustrate how this works, consider the following scenario:

- $A = \{\langle 2, \{x_1 = 3\}\rangle, \langle 3, \{x_2 = 1\}\rangle\}$
- $A' = \{\langle 5, \{x_1 = 2\}\rangle, \langle 4, \{x_2 = 1\}\rangle\}$
- $f(a, a') = a < a'$
- $g(a, a') = a * a'$

$$
\begin{aligned}
g(A, A') &= \{\langle 2, \{x_1 = 3\}\rangle, \langle 3, \{x_2 = 1\}\rangle\} * \{\langle 5, \{x_1 = 2\}\rangle, \langle 4, \{x_2 = 1\}\rangle\} \\
&= \{\langle 2, \{x_1 = 3\}\rangle * \langle 5, \{x_1 = 2\}\rangle, \langle 2, \{x_1 = 3\}\rangle * \langle 4, \{x_2 = 1\}\rangle, \\
&\quad \langle 3, \{x_2 = 1\}\rangle * \langle 5, \{x_1 = 2\}\rangle, \langle 3, \{x_2 = 1\}\rangle * \langle 4, \{x_2 = 1\}\rangle\} \\
&= \{\langle 10, \{x_1 = 3, x_1 = 2\}\rangle, \langle 8, \{x_1 = 3, x_2 = 1\}\rangle, \langle 15, \{x_1 = 2, x_2 = 1\}\rangle, \langle 12, \{x_2 = 1\}\rangle\} \\
&= \{\langle 8, \{x_1 = 3, x_2 = 1\}\rangle, \langle 15, \{x_1 = 2, x_2 = 1\}\rangle, \langle 12, \{x_2 = 1\}\rangle\} \\
&= \{\langle 8, \{x_1 = 3, x_2 = 1\}\rangle, \langle 12, \{x_2 = 1\}\rangle\}
\end{aligned}
$$

In the first step, we expand the cross product of each pair from A and A'. The second step performs the pair-wise operation for every combination (using g to derive the new value). The third step removes the pair $\langle 10, \{x_1 = 3, x_1 = 2\}\rangle$ because it is inconsistent (x_1 cannot take on more than one value). Finally, step four removes the redundant pair $\langle 15, \{x_1 = 2, x_2 = 1\}\rangle$ since it is dominated by $\langle 12, \{x_2 = 1\}\rangle$: $12 < 15$ and $\{x_2 = 1\} \prec \{x_1 = 2, x_2 = 1\}$.

Now that we have a framework for dealing with labelled value sets, all that remains is to augment the traditional algorithms for STP consistency to use the new form of edges. It should be noted that along with the labelled value sets, a global conflict set is maintained (recall that a conflict is an environment that needs to be avoided). In any operation that creates a new labelled value set, the conflict set is consulted and any matching environments (i.e., anything subsumed by a conflict environment) is discarded. The conflicts represent situations that must be avoided, since they have been proven to lead to an inconsistent component STP. They are generated in the augmented all-pairs-shortest-path algorithm when a negative cycle is detected. Algorithm 3 shows the Floyd Warshall's modified algorithm for solving the all-pairs-shortest-path problem.

The subroutine NoEnvironmentLeft checks to see if every possible environment is ruled out by some conflict in S. In such cases, there is no consistent component STP, and thus the DTP is inconsistent. Note that while the original Floyd Warshall algorithm runs in $O(|V|^3)$ time, the variation for labelled distance graphs may be exponential due to the crossproduct of labelled value set operations. The labelled distance graph that results from Algorithm 3 implicitly represents the dispatchable form of every component STP.

We can further continue the analogy to classical STNs by adopting the edge dominance definitions. Rather than only checking a value and dominating an edge in the distance graph, we further check subsumption conditions and dominate pairs in the labelled value set. Consider a triangle of edge value pairs, $\langle a_{AB}, e_{AB}\rangle \in W(A, B), \langle a_{BC}, e_{BC}\rangle \in W(B, C), \langle a_{AC}, e_{AC}\rangle \in W(A, C)$. The pair $\langle a_{AC}, e_{AC}\rangle$ is dominated (and can thus be discarded) if one of the following two conditions hold:

1. Both a_{AC} and a_{BC} are positive, $e_{AB} \cup e_{BC} \prec e_{AC}$, and $a_{AB} + a_{BC} = a_{AC}$
2. Both a_{AC} and a_{AB} are negative, $e_{AB} \cup e_{BC} \prec e_{AC}$, and $a_{AB} + a_{BC} = a_{AC}$

These rules allow us to soundly simplify the labelled distance graph prior to dispatching. When it comes time to dispatch the labelled distance graph, we follow an algorithm analogous to Algorithm 1 that uses labelled value sets for the bounds of an event (rather than single upper and lower bounds). The key difference is that when the bounds are updated (lines 9 and 11 in Alg. 1), we do so over every pair in the labelled value set for that edge, and choose $f \in \{<, >\}$ appropriately to keep the labelled value set for the bounds minimal (i.e., upper bounds use $<$ and lower bounds use $>$). The result is a labelled value set for each bound which is merged with any value sets that already exist for the bounds.

Algorithm 3: Floyd Warshall's All Pairs Shortest Path Algorithm for Labelled Distance Graphs

Input: Labelled distance graph, $\langle V, W, X \rangle$
Output: A dispatchable labelled distance graph with a conflict list S, or INCONSISTENT

```
// Initialize the conflict set S
S = ∅
foreach i ∈ V do
    foreach j, k ∈ V do
        // Use the value set operation with f(a, a') : a < a' and g(a, a') : a + a'
        C_jk = W_ji + W_ik
        if j == k then
            // Check for negative cycles
            foreach ⟨a_i, e_i⟩ ∈ C_jk s.t. a_i < 0 do
                // Add the new conflict environment to S, keeping S minimal
                S = S ∪ {e_i}
                if NoEnvironmentLeft(S) then
                    return Inconsistent
        else
            // Add all of the new value set pairs, keeping W_jk minimal and avoiding conflicts in S
            W_jk = W_jk ∪ C_jk
return ⟨V, W, X⟩, S
```

To select events while dispatching, the strategy employed is to execute an event as early as possible as long as it keeps at least one component STP consistent. The impact of executing each event is tested, and if it causes every component STP to become inconsistent then the execution for that event is delayed. However, no preference is given to events that cause fewer inconsistencies amongst the component STPs.

It is worth noting that the material presented here is a slightly simplified version of [Conrad and Williams, 2011]. In the full description, *activities* are explicitly handled. An activity, in this context, is a process in the DTN (i.e., has a start and end event) that requires the agent to decide on the duration for the activity *when it begins*. In this case, we can no longer choose the end event when we see fit, but still have control over its duration. The true time of an activity may differ (as long as it is within proper bounds), and this assumption leads to the potential of the dispatcher not being able to complete the DTN. This introduction of uncertainty is motivated by the domains the system was constructed for, and is handled primarily by careful bookkeeping during execution.

Contrasting the approaches in [Conrad and Williams, 2011] with the earlier work of [Tsamardinos et al., 2001], the early work explicitly stores information for every complete environment (i.e., component STP). This approach eliminates the need to represent the environment explicitly, but suffers from having to store and work with information for every component STP. In contrast, the compact representation of [Conrad and Williams, 2011] allows for computation to carried out on a number of component STPs simultaneously due to the implicit representation of the labelled value sets.

An alternative method for determining consistency of a DTP has been proposed by [Tsamardinos et al., 2003]. In their work, a meta-CSP is constructed such that every disjunction corresponds to a single variable and every disjunct a value for the meta-CSP variable. Temporal inference is conducted when a variable is set to rule out other variable settings, and the entire search is embedded in a proper CSP solver that allows for nogood learning and domain pruning. Further details are out of the scope of this review.

3.1.3 TCSP

While the techniques presented in the previous section work equally well for TCSPs, as a TCSP is just a particular form of DTP, it is worth mentioning some further work for the sake of completeness. Using ideas introduced in [Shah et al., 2007] for iterative compilation of a TCSP, [Shah and Williams, 2008] introduce a method for compiling and dispatching TCSPs.

There are many similarities between the method for TCSPs, and the previously presented work based on ATMS research. Both attempt to capture the component STPs implicitly to avoid the exponential representation size that appeared in earlier work. Both maintain a set of conflicts, each being a set of choices that cause a temporal inconsistency in the problem. Finally, both store the implication of selecting a particular disjunct or combination of disjuncts. While the two approaches are similar in many regards, and follow from the same motivation, there are some fundamental differences.

The overall approach to determining TCSP consistency can be broken down into four main steps:

1. Construct a single "relaxed" version of the TCSN by turning every disjunctive constraint into a simple temporal constraint by using the largest upper bound and smallest lower bound over all disjuncts in the constraint.

2. Compile the resulting STP into dispatchable form (as per Section 3.1.1).

3. Revert all of the disjunctive constraints back to their original form, and push every disjunct on to a queue for processing.

4. For every disjunct on the queue, select it as the chosen disjunct for whichever disjunctive constraint it appears in, perform a set of local propagation rules until completion, and record the consequences (i.e., bounds tightened) due to the selection.

Of all the steps, number 4 is the most involved. First, we describe the set of propagation rules that are applied until no more updates are possible:

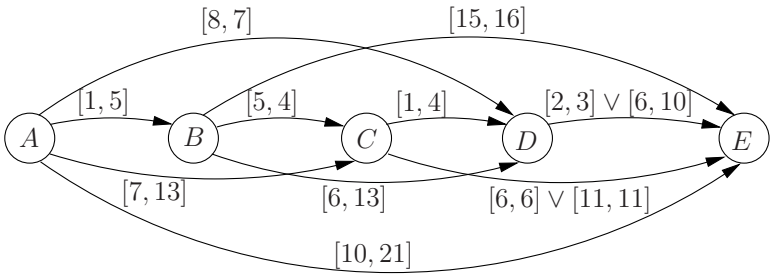
Definition 23 (Dynamic Back-Propagation (DBP) Rules)**.** Given a dispatchable STP with an associated distance graph $\langle V, E\rangle$, we recursively apply the following *dynamic back-propagation rules* to any edge AB that is added or strengthened (i.e., $w(A, B)$ is reduced):

(i) If $A, B, C \in V$, $A \neq B$, $w(A, B) = y > 0$, and $w(B, C) = u \leq 0$, then it follows that $w(A, C) = \min\{w(A, C), y + u\}$.

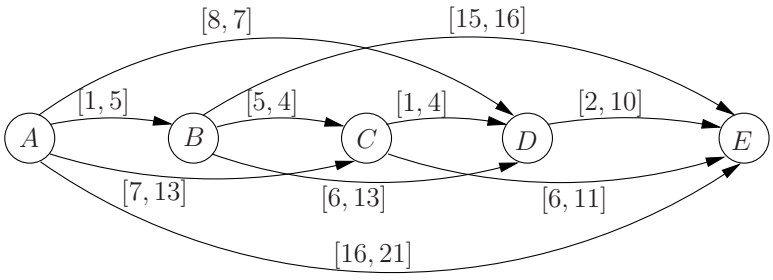
(ii) If $A, B, C \in V$, $A \neq B$, $w(A, B) = x \leq 0$, and $w(C, A) = v \geq 0$, then it follows that $w(C, B) = \min\{w(C, B), x + v\}$.

If we begin with a dispatchable STN, and add or strengthen one of the edges, applying the DBP rules recursively will bring the network back to dispatchable form. Using the DBP rules has the nice property of (possibly) only having to look at a small subset of the STN, as the changes may not propagate through the entire network.

Step 4 of the TCSP approach to consistency applies these rules, and records all of the implied constraints in what is called a *relationship list*. Note, however, that the DBP rules are defined only for STNs. If a disjunctive constraint is encountered during propagation, then each disjunct is treated as a separate case, and the relationship list entries will be predicated on all of the disjunct choices that were assumed. It is during this process that inconsistencies may be discovered, and added to the conflict list. The conflict list is also



(a) Example TCSN.



(b) Relaxed dispatchable form of a TCSN

Figure 14

consulted when considering choices for disjuncts during DBP rules, pruning possibilities that are known to be inconsistent.

To see an example of how the four steps work, consider the TCSP shown in Figure 14a that has two disjunctive constraints: $\{[2,3],[6,10]\}_{DE}$ and $\{[6,6],[11,11]\}_{CE}$. Step 1 replaces them with the two constraints $[2,10]_{DE}$ and $[6,11]_{CE}$, and step 2 produces the relaxed dispatchable form shown in Figure 14b. Step 3 will replace the disjunctive constraints (not shown), and produce a queue containing the four disjuncts: $[2,3]_{DE}$, $[6,10]_{DE}$, $[6,6]_{CE}$, $[11,11]_{CE}$. Finally, step 4 will try each disjunct in the queue, run the DBP rules to completion (for every choice of disjunct found during the propagation as well), and produce a conflict list containing the conflicts $\{[2,3]_{DE},[11,11]_{CE}\}$ and $\{[6,10]_{DE},[6,6]_{CE}\}$. The relationship list constructed for each disjunct will be as follows:

- $[2,3]_{DE} : [13,17]_{AD}, [9,13]_{AC}, [8,9]_{BC}, [12,13]_{BD}, [3,4]_{CD}^{[6,6]_{CE}}$
- $[6,10]_{DE} : [6,10]_{BD}, [8,15]_{AD}$
- $[6,6]_{CE} : [9,9]_{BC}, [1,4]_{AB}, [10,13]_{AC}$
- $[11,11]_{CE} : [5,5]_{BC}, [2,5]_{AB}, [7,10]_{AC}$

Here, we use $[3,4]_{CD}^{[6,6]_{CE}}$ to denote the fact that $[3,4]_{CD}$ is implied only when $[6,6]_{CE}$ is chosen. Note that the other choice for CE, when choosing $[2,3]_{DE}$, caused the conflict $\{[2,3]_{DE},[11,11]_{CE}\}$.

Dispatching proceeds as in previous work – time is allowed to pass until at least one event becomes eligible, and the event is executed as long as doing so does not rule out every component STP. Applicability of an event is checked by setting the time point of the event, and then running the DBP rules to see if any inconsistency is reached. If not, then the event can be safely executed.

Extending the state-of-the-art for TCSP consistency checking in a new direction, [Shah et al., 2009] introduce a method for checking and dispatching TCSNs for multiple agents. The agents are synchronized through "agent occupancy constraints" (limiting the number of activities that an agent can perform). The method supports just-in-time scheduling and agent allocation, and the dispatching is guaranteed to be temporally consistent, logically valid, and maximally flexible. The dispatchable policy is compiled and maintained in an online fashion by using the DBP rules in the same manner as the previous work, but slightly modified to take into account the agent occupancy constraints. The compilation is done in a centralized manner, as supposed to a distributed compilation of the dispatchable form. In Section 4.3 we will see such a procedure for a closely related formalism.

3.2 Strong Controllability

3.2.1 STNU

Introduced in [Vidal and Ghallab, 1996] for a restricted class, and later expanded in [Vidal and Fargier, 1999], determining if an STPU is strongly controllable amounts to rewriting the STPU as a "worst-case" STP, and then checking for consistency. The STPU is strongly controllable iff the STP is consistent.

We first consider the STPU in terms of the linear constraints it represents (similar to checking the consistency of an STP). Recall that every simple temporal constraint, $[l, u]_{XY}$, represents two equations:

$$T_Y - T_X \leq u$$
$$T_X - T_Y \leq -l$$

We transform the STPU into an STP by replacing every constraint that involves an observable event (i.e., from $\mathcal{X}_o$). We thus have four possibilities for the constraint $T_X - T_Y \leq a$:

1. $X, Y \in \mathcal{X}_e$: Both events are executable.
2. $X \in \mathcal{X}_e, Y \in \mathcal{X}_o$: A constraint from an observable event to an executable one.
3. $X \in \mathcal{X}_o, Y \in \mathcal{X}_e$: A constraint from an executable event to an observable one.
4. $X, Y \in \mathcal{X}_o$: A constraint between two observable events.

For the first case there is nothing to modify – with both events executable, all of the uncertainty is removed. For the second case, assume that Y is part of the contingent link $[[l, u]]_{WY}$. In this case, we can rewrite the constraint as follows:

$$\begin{aligned} T_X - T_Y &\leq a \\ T_X - (T_W + \omega_{WY}) &\leq a \\ T_X - T_W - \omega_{WY} &\leq a \\ T_X - T_W &\leq a + \omega_{WY} \end{aligned}$$

Recall that nature will decide the length of ω_{WY}, but we know that at the very least the following constraint holds:

$$0 < l \leq \omega_{WY}$$

We can thus tighten the constraint as follows:

$$\begin{aligned} T_X - T_W &\leq a + \omega_{WY} \\ \therefore T_X - T_W &\leq a + l \end{aligned}$$

This constraint captures the worst case scenario that can occur due to nature's choice of the length of the contingent WY constraint, but involves only executable events. We can follow a similar process for case 3, where X is observed and Y is executable (assume now that X is part of the contingent link $[[l, u]]_{ZX}$):

$$\begin{aligned} T_X - T_Y &\leq a \\ (T_Z + \omega_{ZX}) - T_Y &\leq a \\ T_Z + \omega_{ZX} - T_Y &\leq a \\ T_Z - T_Y &\leq a - \omega_{ZX} \\ \therefore T_Z - T_Y &\leq a - u \end{aligned}$$

Note that we use the upper bound here, since it provides the tightest restriction of the constraint. Finally, we combine both approaches to handle case 4 (assuming that X and Y are part of the contingent links $[[l_{ZX}, u_{ZX}]]_{ZX}$ and $[[l_{WY}, u_{WY}]]_{WY}$ respectively):

$$\begin{aligned} T_X - T_Y &\leq a \\ (T_Z + \omega_{ZX}) - (T_W + \omega_{WY}) &\leq a \\ T_Z - T_W + \omega_{ZX} - \omega_{WY} &\leq a \\ T_Z - T_W &\leq a - \omega_{ZX} + \omega_{WY} \\ \therefore T_Z - T_W &\leq a - u_{ZX} + l_{WY} \end{aligned}$$

We now have a way to rewrite every constraint that involves an observed event into one that that does not. The result is a set of constraints between only executable events: an STN. The resulting STN is consistent if and only if the original STPU is strongly controllable, and any valid execution of the events in the STN is a valid execution of the events for the STPU. Since the rewriting of the constraints is linear, and verification of STP consistency is polynomial, determining if an STPU is strongly controllable is a polynomial task.

3.2.2 DTNU

While it is tractable to check for strong controllability of STPUs, the task is considerably more difficult when disjunctions are introduced. In [Peintner et al., 2007], a method for determining if a DTNU is strongly controllable is presented, and a number of special cases of DTPUs are shown to be equivalent to DTPs. In general, however, the complexity of checking for strong controllability is in the class EXPSPACE.

The idea behind dealing with a DTPU is to use the check of strong controllability in an STPU within the framework of a meta-CSP search. The variables of the meta-CSP correspond to the disjunctions in the DTPU, and the values correspond to the specific disjunct that is to be satisfied. Search is subsequently conducted over all of the meta-CSP variables, but done so in a particular order according to the type of disjunction (i.e., (mixed) executable DTNU, contingent DTNU, or otherwise).

The search begins by selecting a disjunct for every simple DTN constraint (i.e., a disjunction of simple temporal constraints). They are selected recursively in the meta-CSP, and selections are undone if and when a temporal inconsistency is found. Once a setting to every DTN constraint is found, the next phase of variable selection begins, and the meta-CSP searches through the setting of disjunctive contingent constraints. If just one of these variable selections causes an inconsistency, then the procedure backtracks to the first phase, as *every* selection of disjunctive contingent constraints must work. If successful (i.e., a consistent setting to the disjunctive contingent constraints is found), then the third and final phase is begun.

The third phase consists of searching through the (mixed) executable DTNU constraints for *every* consistent setting. The second phase maintains the intersection (over all disjunctive contingent constraint choices) of consistent settings to the (mixed) executable DTNU constraints. The reasoning for this approach is that we need at least one consistent setting to the (mixed) executable DTNU constraints that works for *every* setting of the disjunctive contingent constraints. This situation will happen if and only if the intersection ends up being non-empty. If, at some point, the intersection does become empty, the second phase backtracks once again for a new setting in the first phase of the meta-CSP search.

It should be noted that temporal consistency is verified differently at each phase. In the first phase, only a subset of the constraints have been selected, and so the induced STP is checked for consistency. In the second and third phases, however, we have selections to disjunctive contingent constraints, which yields an STPU that must be checked. Here, the previously introduced method for STPU strong controllability is used.

Depending on the presence of certain types of constraints, five cases are identified and investigated further [Peintner et al., 2007]. We review each of them here, and use the following interpretation for variables in the complexity notation:

- n: Number of events.
- k: Maximum number of disjuncts in a constraint.
- S: Number of single-disjunct constraints, plus disjunctions containing only simple temporal constraints.
- C: Number of disjunctive contingent constraints.
- E: Number of (mixed) executable DTNU constraints.

Case 1: *STPU*

The first simple case is when every constraint has only one disjunct. Here, the DTPU is obviously equivalent to the STPU problem, and the technique for strong controllability of STPUs can be used (see Section 3.2.1).

Case 2: *Totally Simple*

In the second case, consider when all of the disjunctive constraints present in the DTNU are over simple temporal constraints. In this case, uncertainty is present only in single disjunct constraints. Here, we need only to find a single setting of the disjunctive constraints such that the induced STPU is strongly controllable. This assumption means we can stop searching through the disjunctive constraints once a single consistent setting is found. The complexity in this case reduces to that of solving consistency of a DTP: $O(n^2Sk^S)$.

Case 3: *Simple-Nature-Dependent*

This case arises when there is at least one disjunctive contingent constraint, but no (mixed) executable DTNU constraints. It is called simple-nature-dependent because it is nature that will decide which disjunct

in the DTNU constraints will be satisfied (as they are all disjunctive contingent constraints). The complexity for this variation is $O(n^2Sk^S + n^2Sk^Ck^S)$ (note the added k^C to the second term due to searching through the disjunctive contingent constraints).

Case 4: *General DTPU*

In the most general case, we have constraints of every type. In the worst case, the procedure will need to check $k^Sk^Ck^E$ disjunct combinations. This result brings the complexity of the 3-phase meta-CSP search to $O(n^2Ek^Sk^Ck^E + n^2k^SSk^{Sk^C})$.

Case 5: *Simple-Nature*

In the final case, there are no disjunctive contingent constraints. This assumption means that all of the contingent constraints are simple. Nature no longer has control over which disjuncts are selected. As such, the complexity above is reduced with $C = 0$ to be $O(n^2k^S(Sk^S + Ek^E))$.[6]

While not mentioned in the literature, there is a correspondence between the (mixed) executable DTNU constraints and the DTN constraints that have only simple temporal constraints as disjuncts. There is a common trick employed to have consecutive contingent constraints compiled away in an STNU – the shared event is split into two nodes and connected by the simple temporal constraint $[0, 0]$. A similar technique can be employed for any executable STNU constraint to turn it into an executable STN constraint. While this technique introduces new nodes, it provides a more concise formalism to work with. Compiling away the executable STNU constraints would create at most E new nodes, and shift the disjunctions to be entirely simple in nature.

3.3 Weak Controllability

3.3.1 STNU

Since the task of weak controllability of an STNU is not so applicable in practice, there has been little emphasis on techniques to solve it in the literature. [Vidal and Fargier, 1999] provide an algorithm for checking weak controllability, and they conjecture that the task is co-NP-Complete. The premise behind the approach for checking weak controllability of an STNU is that one needs only to inspect the extremities of the contingent links – any duration in between will automatically follow. This property leads to the simple search procedure shown in Algorithm 4.

The method NOTCONSISTENT attempts to check for STP consistency. If inconsistent, or if any contingent link is tightened as part of the propagation, then true is returned. STRONGLYCONTROLLABLE represents a polynomial procedure that checks for strong controllability of the STNU (recall that strong controllability implies weak controllability). The check for strong controllability also serves as the base case – when there are no contingent links, and the network is consistent, then it is trivially strongly controllable. Line 5 selects a contingent constraint arbitrarily and this part is the branching point for the algorithm. We first try the lower bound (line 6), and if that passes we try the upper bound as well (line 8).

Similar to CSP techniques that ensure k-consistency of the constraints, an extension suggested by [Vidal and Fargier, 1999] replaces the NOTCONSISTENT method: *k-weak controllability*. For $k = 2$ or $k = 3$, a stronger form of propagation is used and allows for backtracking far sooner in the search process.

[6] Note that [Peintner et al., 2007] state the complexity as $O(n^2k^S(S + Ek^Sk^E))$, but this analysis appears to be erroneous.

Algorithm 4: WEAKLYCONTROLLABLE: Verify weak controllability of an STNU.

Input: STNU, $(\mathcal{X}_e, \mathcal{X}_o, C_f, C_c)$
Output: true / false if the network is weakly controllable.

if NOTCONSISTENT$(\mathcal{X}_e, \mathcal{X}_o, C_f, C_c)$ **then**
  return false

if STRONGLYCONTROLLABLE$(\mathcal{X}_e, \mathcal{X}_o, C_f, C_c)$ **then**
  return true

$[[l, u]]_{XY} \leftarrow$ some contingent constraint in C_c

if *not* WEAKLYCONTROLLABLE$(\mathcal{X}_e, \mathcal{X}_o, C_f \cup \{[l, l]_{XY}\}, C_c \setminus \{[[l, u]]_{XY}\})$ **then**
  return false

return WEAKLYCONTROLLABLE$(\mathcal{X}_e, \mathcal{X}_o, C_f \cup \{[u, u]_{XY}\}, C_c \setminus \{[[l, u]]_{XY}\})$

3.3.2 DTNU

Following on the insights for determining weak controllability of an STNU, [Venable et al., 2010] introduced two methods for determining if a DTNU is weakly controllable. The first works for a restricted class of DTNUs (Simple Natured) while the second is an exhaustive approach that has a larger complexity, but is more widely applicable.

When the disjunctive constraints are limited to not contain any contingent links, then we can take advantage of the fact that if any component STPU (a choice of disjunct for every disjunctive constraint) is weakly controllable, then so is the DTPU. Similar to the procedure for determining weak controllability of an STNU, the approach for DTNUs need only to investigate the boundary projections in a systematic manner. The approach taken for weak controllability of a Simple Natured DTNU is to search through the boundary projections recursively, and at the leaf nodes (when no contingent constraints remain), the DTP is solved for all of the valid solutions (i.e., consistent component STPs). Upon backtracking, the intersection of solution sets are checked: if non-empty, then we continue to backtrack; otherwise the contingent link is partitioned and checked to ensure that both sides are weakly controllable.

It should be noted that due to the process of splitting the contingent link upon backtracking, there must be an implicit assumption that the duration for any contingent link will be an integer between the lower and upper bounds (inclusive). When we allow disjunctions of contingent links (i.e., a general DTNU), then the method proposed is an exhaustive search that enumerates every possible setting to the contingent links (again, finite domain of link duration is assumed). In an initial phase, the contingent links are treated as regular simple temporal constraints, and every potential solution is computed for the corresponding DTN. Next, a hash is created that maps the value of a contingent link to the set of solutions that respect that value. This mapping is used in the final phase, where we systematically search through every possible setting to the contingent constraints, ensuring that the solution set that corresponds to the intersection of the mapping for every contingent constraint setting is non-empty.

Since the first method potentially splits the contingent constraints, both methods end up being exponential in the number of the contingent constraints that exist as well as exponential in the number of disjunctive constraints that exist. However, the authors conjecture that since a weakly controllable STNU component may arise frequently in practice, the first method has a greater opportunity to perform potentially far less computation.

3.4 Dynamic Controllability

3.4.1 STNU

As we have previously discussed, determining the dynamic controllability of an STNU is one of the most studied problems in the field of plan dispatchability. With over ten years of publications on the topic, the approaches have become progressively more efficient. Here, we give a brief perspective on the improvements that have led to the state-of-the-art, and present the modern techniques for asserting dynamic controllability of an STNU.

When dynamic controllability was introduced in [Vidal and Fargier, 1999], the authors conjectured that the complexity was in the class of PSPACE problems. The motivation for this conjecture was due to the "agent versus nature" aspect of the dynamic controllability problem. The approach suggested then was to use a classical alpha-beta scheme for two-player games. One player is the agent, choosing the times of the executable events, and the other player is nature, choosing the durations of the contingent links.

The next approach took the unique strategy of using Timed Game Automaton (TGA) to compile and dispatch the plan for dynamic controllability [Vidal, 2000]. The approach follows from the previous motivation of having the agent versus nature and turn-taking in a temporal setting. Standard timers that are associated with TGA's (renamed to *stopwatches* in this work) are used as guards in the network to handle contingent links as well as free constraints. A count-down timer is introduced, called an *egg timer*, that is used to handle a special form of free constraint where the duration of the link must be chosen when the start event is scheduled. The appeal of this approach is the possible extensions that a TGA framework allows for: generalized conditional planning, combining a number of preprocessed plans, synchronizing constraints, etc.

The TGA is built by using a form of regression, and ends up enumerating every possible scenario for the choice of agent control and nature's choices. This approach may lead to a combinatorial explosion, but there are a few key points that were introduced here and carried on in later research. One is the notion of a "wait" constraint; the agent must wait until a specified time before acting. The other is the notion of "waypoints"; when the plan can be effectively partitioned in a temporal fashion, each partition is handled individually and combined through the waypoints.

The next entry into the array of research for dynamic controllability saw the introduction of sufficient conditions on an STNU that indicates if it is dynamically controllable [Morris and Muscettola, 2000]. In this work, they present two methods for dispatching STNUs: one that retains maximal flexibility at the cost of online computation, and another that sacrifices flexibility for greater efficiency in dispatching. Both rely on the notion of boundary projections:

Definition 24 (Boundary Projection)**.** For an STNU, $\mathcal{N}$, a *boundary projection* is a projection $\mathcal{N}_\omega$, such that every contingent length $\omega_{X_iY_j} \in [[l, u]]_{X_iY_j}$ is set to be either l or u.

The two methods presented in [Morris and Muscettola, 2000] are the first that make use of the boundary projections: many algorithms on the network (e.g., shortest path algorithms) will work for every projection if and only if they work for every boundary projection.

The first method involves the use of tags for propagating the bounds according to every possible boundary projection. As an initial step, every boundary projection is processed for STN consistency, and any edges that are tightened are labelled with the new value and tag that caused the tightening. The process is similar to the use of DBP rules described in Section 3.1.3. The tags are propagated during dispatching, and the approach retains maximal flexibility. The problem, however, is that many edges can no longer be removed by dominance rules. An edge is now dominated if and only if it is dominated in *every* boundary projection

(which occurs rarely in practice). This leads to the adaptation of Algorithm 1 requiring a large amount of propagation every time an event is executed.

The second approach considers a sufficient condition for efficient dispatchability of a dynamically controllable STNU: when the only propagations to an observable event while dispatching are through the contingent link for that event. The property can be formally defined as follows:

Definition 25 (Network Safety)**.** Consider the STN that is formed by replacing every contingent link in an STNU $\mathcal{N}$ with a simple temporal constraint of the same bounds. For the distance graph of the STN, a *tight edge* (X, Y) is an edge such that no shorter path from X to Y exists (i.e., the weight of (X, Y) is the same in both the distance graph and associated d-graph). $\mathcal{N}$ is a *safe network* if and only if the following holds:

1. The forward and reverse edges in the distance graph that originated due to a contingent link are tight edges.

2. Any positive edge (X, Y) in the distance graph that is due to a contingent link dominates all non-negative edges whose destination is Y.

3. Any negative edge (X, Y) in the distance graph that is due to a contingent link dominates all negative edges whose source is X.

The first item is necessary for an STNU to even be weakly controllable (if a contingent edge is squeezed, then there is a way for nature to make the network inconsistent no matter what the agent does), while the other two ensure that all propagations go through the contingent link while dispatching. It is noted that safe networks are likely to be rare in practice, but one may be able to turn an unsafe network into a safe one by strengthening certain free constraints. Doing so comes at the loss of flexibility, but allows us to retain efficient dispatchability without the combinatorial blowup that using tags would entail.

An STNU that is not safe but can be made safe by strengthening the free constraints is called *potentially safe*. Unfortunately, [Morris and Muscettola, 2000] showed that determining if an STNU is potentially safe is an NP-hard problem, and a network may be dynamically controllable and *not* potentially safe. However, the methods introduced by [Morris and Muscettola, 2000] led to the insights in following research.

The first polynomial algorithm for dynamic controllability of an STNU was presented in [Morris et al., 2001]. The paper describes an approach for making an unsafe network dispatchable. The unsafe network must be *pseudo-controllable*, meaning that no contingent link is tightened when treating the STNU as an STN and checking for consistency, and they check this necessary condition for dynamic controllability before performing their main modification to the network: triangular reductions.

The all-pairs-shortest-path algorithm is used to compute the d-graph of the pseudo-controllable STNU treated as an STN. From this point, further reductions occur in the form of a set of rules that operate over a triangle of edges. While the precise details are out of the scope of this review (similar rules will be introduced for later approaches), it is worth giving an intuition for the approach taken. Due to the uncertainty introduced by contingent links, a new constraint is introduced that allows for effective dispatch; the *wait constraint*.

Definition 26 (Wait Constraint)**.** The *wait constraint* $\langle Z, t\rangle$ on the edge (X, Y) in the network indicates that Y cannot be executed until either t time as passed after X is executed, or the event Z is executed.

Wait constraints are introduced by analyzing triples of events in the d-graph, and a procedure called 3DC propagates these wait constraints around the network soundly until a fixed point is reached (possibly removing some wait constraints that are redundant). The resulting network may contain wait constraints, but they can be handled easily in a modification to Algorithm 1 that only enables events when all of the

appropriate wait constraints are satisfied. Due to the propagation that must occur, the complexity of the algorithm is $O(n^5)$, and only pseudo-polynomial – the complexity analysis assumes that the maximum value of any contingent link is finitely bounded (i.e., a higher potential length of a contingent bound may increase the complexity). Despite this drawback, the work represents the first polynomial algorithm for the problem of dynamic controllability, and is guaranteed to produce a network that is maximally flexible and minimal over all such networks.

Continuing along the same line of research, [Morris and Muscettola, 2005] presented a variation of the approach for dynamic controllability that has the same complexity ($O(n^5)$), but is *strongly polynomial* (i.e., no longer requires a low bound on the contingent link duration). Aside from removing the pseudo-polynomial restriction, this work presents the update rules that use wait constraints in a more unified and intuitive manner. Rather than regressing wait constraints throughout the completed d-graph, the update rules are succinctly formalized as label updates on the STN's distance graph (recall that we convert the STNU into an STN by treating the contingent links as simple temporal constraints). The key behind the complexity analysis is using the Bellman-Ford cutoff to argue that only a quadratic number of iterations are required.

Not long after the first strongly polynomial algorithm for dynamic controllability of an STNU was published, one of the primary authors extended the approach with a new cutoff algorithm to produce a check for dynamic controllability of an STNU in $O(n^4)$ time [Morris, 2006]. The update rules were once again simplified, and we present the main details of their approach. The general idea is to assert that a particular type of negative cycle does not exist in the network. While the paper presents a method for determining *if* an STNU is dynamically controllable, they only conjecture at the existence of a dispatching algorithm (which is partially dealt with in [Hunsberger, 2010] as we see below).

Rather than perform an all-pairs-shortest-path, the network is augmented with the labels needed for contingent links, and updates are done iteratively until a cutoff bound is reached. Before describing the rules for updating, we introduce some terminology for the network that contains contingent link labels:

Definition 27 (Contingent-Labelled Distance Graph)**.** Given an STNU $\mathcal{N} = (\mathcal{X}_e, \mathcal{X}_o, C_f, C_c)$, the *contingent-labelled distance graph* $\langle V, E\rangle$ is formed by taking the nodes as $V = \mathcal{X}_e \cup \mathcal{X}_o$, and the edges as $E = E_f \cup E_c \cup E_l$, where:

- $E_f = \{X \xrightarrow{u} Y, X \xleftarrow{-l} Y \mid \forall [l,u]_{XY} \in C_f\}$
- $E_c = \{X \xrightarrow{u} Y, X \xleftarrow{-l} Y \mid \forall [[l,u]]_{XY} \in C_c\}$
- $E_l = \{X \xrightarrow{y:l} Y, X \xleftarrow{Y:-u} Y \mid \forall [[l,u]]_{XY} \in C_c\}$

The graph is in fact a multi-graph, as there may be multiple edges between the same two events. Edges in E_f are the standard upper bounds associated with the distance graph of an STN, and E_c are the similar bounds, but for the contingent constraints. The new addition here are edges of the form $X \xrightarrow{z:n} Y$ and $X \xrightarrow{Z:n} Y$ (respectively called *lower-case* and *upper-case* edges). The upper-case value represents the case where nature decides the constraint should take on the maximum length, and the lower-case value represents the case where it takes on the minimum length. To connect this notation with prior work, the wait constraint $\langle Z, t\rangle$ between events X and Y is represented by $X \xleftarrow{Z:-t} Y$.

The labels provide a concise framework for updating values in the contingent-labelled distance graph through the following four reduction rules and one label-removal rule:

- (UPPERCASE): If $X \xleftarrow{W:m} Y \xleftarrow{n} Z$, then add $X \xleftarrow{W:(m+n)} Z$

- (LOWERCASE): If $X \xleftarrow{m} Y \xleftarrow{y:n} Z$, then add $X \xleftarrow{m+n} Z$ (only if $m \leq 0$ and $X \neq Y$)
- (CROSSCASE): If $X \xleftarrow{W:m} Y \xleftarrow{y:n} Z$ then add $X \xleftarrow{W:(m+n)} Z$ (only if $m \leq 0$ and $W \neq Y$)
- (NOCASE): If $X \xleftarrow{m} Y \xleftarrow{n} Z$, then add $X \xleftarrow{m+n} Z$
- (LABELREMOVAL): If $X \xleftarrow{x:m} Y \xleftarrow{X:n} Z$, then add $Y \xleftarrow{n} Z$ (only if $n \geq -m$)

All of the rules infer new edges based on the labelled edges that already exist. Note that only upper-case edges are propagated (through UPPERCASE and CROSSCASE reduction rules), and lower-case edges will never change. Thus, there will only ever be $|C_c|$ lower-case edges. Also, upper-case edges will always point to the start event of the contingent link. Finally, note that the NOCASE rule is simply the one that would be used in an algorithm such as Floyd-Warshall's for finding the all-pairs-shortest-path.

The key behind the approach for checking dynamic controllability is to repeatedly apply the above rules in a structured manner until a sufficient number of iterations have occurred. To describe the structured manner of iteration, and the stopping condition, we introduce further terminology.

Definition 28 (AllMax Projection)**.** The *AllMax projection*, $\mathcal{N}_{max}$, is the boundary projection of STNU $\mathcal{N}$ where every contingent link $[[l, u]]_{XY}$ is replaced by the simple temporal constraint $[u, u]_{XY}$.

The AllMax projection is used to continuously check for consistency in the dynamic controllability check presented below. If it ever becomes inconsistent, then the STNU is not dynamically controllable. Alternatively, if we iterate a sufficient number of times and $\mathcal{N}_{max}$ remains consistent, then the STNU *is* dynamically controllable. Generating $\mathcal{N}_{max}$ is done repeatedly throughout the process, so it helps to view it as the contingent-labelled distance graph with the lower-case edges deleted, and upper-case edges turned into regular edges (keeping only the lowest weighted edge between two events). To describe the algorithm, we require some further definitions that involve reductions that are possible on a path of edges in the contingent-labelled distance graph:

Definition 29 (Path Reductions)**.** The *reduced distance* of a path in the contingent-labelled distance graph is the sum of the edges in the path, disregarding the distinction between upper-case, lower-case, and regular edge types. We say that $\mathcal{P}$ is a *reducible path* if it contains a pair of edges to which one of the edge-reduction rules can be applied. Replacing the pair of edges gives us a new path, $\mathcal{P}'$, and we say that $\mathcal{P}'$ is a *reduction* of $\mathcal{P}$. We further define a path to be *reducible* if it can be reduced to a single edge through a sequence of reduction rules. Finally, a path is *semi-reducible* if it can be reduced to a path that is free of lower-case edges.

By inspecting the reduction rules above, we can see that every one of them preserves the reduced distance of a path. The reductions can be seen as gradually transforming the reduced distance in the contingent-labelled distance graph into normal distance in the AllMax projection. A fundamental result is introduced for this framework: *An STNU is dynamically controllable if and only if it does not have a semi-reducible negative cycle.*

From this point, an $O(n^5)$ algorithm is possible, but further analysis allows for a greater efficiency. The algorithm proposed indeed looks for a semi-reducible negative cycle, but does so very carefully: it suffices to focus on what is referred to as an *extension sub-path*.

Definition 30 (Extension Sub-paths)**.** Assume that e is the lower-case edge $X \xrightarrow{y:n} Y$, within some path $\mathcal{P}$. The *extension* of e is the sub-path $\mathcal{P}_e$ where the following holds:

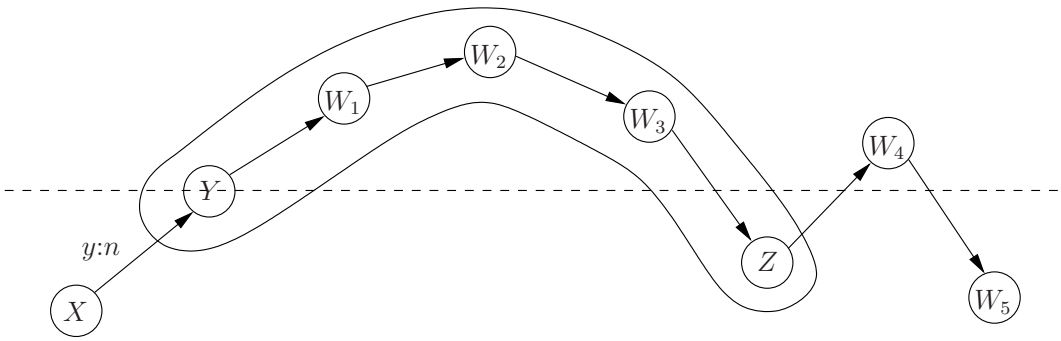


Figure 15: Example of an extension subpath for the lower-case edge $X \xrightarrow{y:n} Y$.

- The first node in $\mathcal{P}_e$ is Y, and ends at some node Z.
- The reduced distance from Y to Z in $\mathcal{P}_e$ is less than or equal to 0.
- The reduced distance from Y to any other node in $\mathcal{P}_e$ is positive.

Using a visualization technique introduced in [Hunsberger, 2010], Figure 15 demonstrates the extension sub-path of the lower-case edge $X \xrightarrow{y:n} Y$. The height represents the reduced distance from Y. Notice that the edge leading to Z is the first such edge in the path where the reduced distance from Y to the end-point of the edge is less than (or equal to) 0. This edge is referred to as a *moat edge*, and it plays a role in determining if a path is semi-reducible. We further quantify a moat edge by saying that it is *unusable* if it is an upper-case label that corresponds to the lower-case label at the start of the extension (i.e., $W \xrightarrow{Y:m} Z$ is the moat edge for the sub-path extension for $X \xrightarrow{y:n} Y$). The following property gets us one step closer to the sufficient condition for the algorithm:

A path $\mathcal{P}$ is semi-reducible if and only if every lower-case edge in $\mathcal{P}$ has a usable moat edge in $\mathcal{P}$.

An interesting corollary to this property is that in a semi-reducible path there must be a canonical elimination of the lower-case edges through reduction, by reducing the inner-most semi-reducible paths and working outwards. We say that a semi-reducible path for edge $X \xrightarrow{y:n} Y$ is *breach free* if there is no edge in the path with the label $Y{:}m$, and we say the repetition of a lower-case edge in a semi-reducible path is *nested* if the extension from one occurrence contains the other. This definition leads us to the following property:

If an STNU has a semi-reducible negative cycle, then it has a breach-free semi-reducible negative cycle with no nested repetitions.

This property allows us to focus our search on the canonical elimination of semi-reducible sub-paths until either an inconsistency in the AllMax projection is found, or we have sufficiently exhausted the elimination. Recalling that we can never add a lower-case edge by the rules above, there is a maximum of $|C_c|$ reductions that must occur since there greatest depth of nesting will be $|C_c|$. The results culminate to give us Algorithm 5 that takes advantage of the above properties.

In the algorithm, the inner loop will propagate forward from every lower-case edge looking for a moat edge before running into a breach, and only on those paths that are semi-reducible. In such situations, repeated application of the NoCase, LowerCase, and CrossCase will allow us to add an edge between X

Algorithm 5: Algorithm to check an STNU for Dynamic Controllability

Input: STNU, $\mathcal{N} = (\mathcal{X}_e, \mathcal{X}_o, C_f, C_c)$
Output: true or false depending on if $\mathcal{N}$ is dynamically controllable.

```
for i = 1 ··· |C_c| do
    if N_max is inconsistent then
        return false
    foreach lower-case edge e = X --y:n--> Y do
        foreach breach-free semi-reducible extension of e starting at Y and ending with Z do
            Add a reduced constraint from X to Z
if N_max is inconsistent then return false;
else return true;
```

and Z with the reduced distance (either as a regular edge or an upper-case edge). This new edge essentially encodes the information of the lower-case edge into an edge that can be used for the AllMax projection.

If the Algorithm 5 completes, and the AllMax projection remains consistent throughout, then the original STNU is guaranteed to be dynamically controllable. Unlike most of the prior approaches, however, the final product (i.e., the contingent-labelled distance graph after $|C_c|$ iterations) is *not* in dispatchable form. In [Morris, 2006], an algorithm was conjectured with a time complexity of $O(n^4)$ that acts as a pre-processing step once the network is known to be dynamically controllable, but further details were not given. To rectify this issue, [Hunsberger, 2010] proposed an *online* algorithm that accepts the processed contingent-labelled distance graph from Algorithm 5, and dispatches the network while amortizing the $O(n^4)$ time over the execution that takes place.

The most expensive portion of the algorithm is re-establishing the consistency of the AllMax projection. This part of the algorithm may require $O(n^3)$ time, but can be done while the agent is waiting for time to pass (either to satisfy a lower bound or to wait for an observed event to occur). The significant difference to the proposed algorithm is that it breaks from the traditional method of generating every implied constraint prior to dispatching. Instead, the required constraints are computed on the fly.

To effectively dispatch the network, a set of *core edges* are identified. These are precisely the ordinary and upper-case edges from the network produced by Algorithm 5. Two networks are maintained during the dispatching process: one that contains all core edges (equivalent to the AllMax projection), and one that contains only the ordinary edges. Every time that an event is executed, both of these networks are updated (which introduces the $O(n^3)$ complexity for using Floyd Warshall's algorithm).

The key behind the online updating relies on the connection between upper-case edges and the previously defined wait constraints. The upper-case edge $X \xrightarrow{Z:-n} Y$ represents the fact that Y cannot be executed until n time has passed after X is executed, or Z (an observed event) has happened. In the situation where Z has not yet occurred while dispatching, we may simply regard the XY edge as the ordinary link $X \xrightarrow{-n} Y$, and indeed that is what the AllMax projection effectively does. If Z is observed to have occurred, however, the dispatcher removes the upper-case edge from the set of valid upper-case edges, and subsequent computations of the AllMax projection will not contain the $X \xrightarrow{-n} Y$ restriction.

Thus, the general idea behind the new dispatching algorithm is to maintain the AllMax projection with upper-case edge information (treated as ordinary constraints), until observed time points are executed. With this approach, the AllMax projection will contain all of the information needed to avoid an inconsistency

with the contingent links. The $O(n^3)$ time required to update the matrices may seem prohibitive, but the time bounds for the events to be executed next can be computed prior to running the updates. When the dispatcher is waiting for time to pass (either for a lower bound on the next executable event, or a contingent event to occur), it is able to re-establish the consistency in the two networks.

As pointed out earlier, the majority of STNU dynamic controllability literature was predicated on a slightly incorrect definition about how dynamic execution strategies are constructed. In [Hunsberger, 2009], this inconsistency in the formalism was fixed, and showed that subsequent techniques were not affected by the problem in the definition. Additionally, [Hunsberger, 2009] provided a new characterization of dynamic execution strategies that is based on *real-time decisions*. We review this contribution here in brief, as it is the most recent formalism of the dynamic controllability problem for STNUs.

Rather than defining strategies as a mapping from complete situations to complete schedules, Hunsberger considers the viewpoint of partial schedules and focuses on the decisions an agent must make based purely on the timing of events that have occurred in the past. Two types of execution choices are introduced:

Definition 31 (Real-time Execution Decision)**.** A *real-time execution decision* (RTED) represents an action that an agent may take while dispatching a network that is dynamically controllable. It can be one of two things: WAIT or $(\tau, \mathcal{X})$. The semantics for the WAIT decision is that the agent waits until some observed event occurs (i.e., just wait for something to happen). The semantics for the $(\tau, \mathcal{X})$ decision is to wait τ time, and if nothing happens before then, execute the events in the set $\mathcal{X}$.

The WAIT decision is used in a situation where the remainder of the network depends on the timing of events that are yet to be observed. The $(\tau, \mathcal{X})$ decision is inspired by the wait constraint we have seen previously – if some observed event were to occur, then the agent registers which events have occurred and selects a new decision based on the schedule executed thus far. Instead of a dynamic execution strategy, dynamic controllability is characterized in terms of a *RTED-based strategy* (RTEDS) that maps a partial schedule to an RTED. To show correctness, a correspondence is established between the dynamic execution strategy of a dynamically controllable STNU and an RTEDS.

It should be noted that the new semantics are demonstrated by a reformulation of the $O(n^5)$ dispatching approach of [Morris et al., 2001], rather than the more recent approach of [Morris, 2006]. Instead of describing the dynamic execution strategy function, [Morris et al., 2001] give a procedure to iteratively generate a schedule. This procedure is adapted in [Hunsberger, 2009] to use the new semantics (i.e., the WAIT and $(\tau, \mathcal{X})$ decisions), and allows for a much more intuitive exposition of what decisions the agent should be making during the dispatch of an STNU that is dynamically controllable.

3.4.2 TCSNU

In [Venable et al., 2010], a method is proposed for dynamic controllability of a TCSNU. Since the method relies on a canonical ordering of the constraints in a disjunction such that each interval is larger than the previous, the authors only consider dynamic controllability of a TCSNU, and not the more general DTNU. The method is based on the first (pseudo) polynomial version of dynamic controllability for STNUs, and mirrors many of the concepts introduced in that work [Morris et al., 2001].

To accomplish a procedure that follows the same principles as those in [Morris et al., 2001], the wait constraint is generalized to include a disjunction of intervals. The semantics of the wait constraint $\langle Z, b, \tau \rangle$, which appears between events X and Y, is that "Y cannot be executed until either Z is executed, or t time has passed since X has executed and t is in some interval of the disjunction τ". Here, b is simply the minimum over all lower bounds in τ, and is maintained for notational convenience.

Like the wait constraints described in [Morris et al., 2001], the wait constraints introduced in [Venable et al., 2010] are generated for certain triples of events, and then propagated around the network until a fixed point is reached. There are a number of conditions that may arise during the propagation that indicate the DTNU is not dynamically controllable, but if the procedure completes then we are guaranteed that it can be dispatched. Further, wait constraints may remain in the network and dispatching with these are similar to the STNU case – they serve as further conditions on when an event can become enabled.

A recent extension to the state of the art for DTN consistency and dispatching allows for contingent edges to be included [Conrad and Williams, 2011]. At the time of writing the details are preliminary, but the idea builds on the framework of ATMS inspired labels to permit the use of algorithms analogous to those found in [Morris et al., 2001]. This approach is made possible by the general nature of the labelled value sets, and the flexibility of the use of operators over labelled edges that we described in Section 3.1.2.

4 Extensions

4.1 Resources

In the scheduling community, dealing with temporal constraints is not the only issue that is typically considered: resource use also plays an important role in many applications. This motivation is behind the addition of resource requirements in a plan dispatchability setting. [Wallace and Freuder, 2005] presents a formalism that allows for resource constraints to be specified, and investigates the issues surrounding the effective dispatch of a network that has associated resource constraints.

In this first iteration of the work, the dispatching may not be maximally flexible, but it at least guarantees the dispatchability of a network. The resources are introduced as consumable entities that deplete or increase linearly over time when an action is being performed (an action is represented by a pair of events in the STN), and the focus of the paper is to handle the case of a *bout*: a period of time where the resource either monotonically decreases or monotonically increases.

The resource constraints for an activity are modelled using an auxiliary network referred to as the *consumable resource network* (cRN). There is one cRN per resource, and every activity that uses the resource is represented by links from the STN to the cRN indicating the multiplicative factor of resource use per unit time (recall that resource use is a linear function). All of the events in a cRN are tied together with a single inequality that enforces the capacity of the resource. With this formalism, the question of dispatchability is modified to not only executing the events while avoiding temporal inconsistencies, but also to ensure that the resource bound is never exceeded. [Wallace and Freuder, 2005] are concerned with handling just a single resource, but the methods employed are easily extendible to multiple resources with independent bouts.

Example 6 (Single Resource STN)**.** An example STN, with an associated cRN for a single resource, is given in Figure 16. Note that the bounds for the resource use match the time bounds of the associated event, after applying the correct multiplicative factor (e.g., $[2, 4]_{X_1Y_1}$ in the STN corresponds to $[10, 20]$ in the cRN with the factor of 5). If n activities use a resource, then the constraint in the cRN will be an n-ary constraint (not simply binary).

Much of the focus for handling resources in an STN involves identifying sufficient conditions for the resource constraints to be violated or trivially satisfied. An example of the latter is when the sum of the upper bounds on the resource use is less than the resource's capacity – while this fact may not be true at the start of dispatching, it may become true when upper bounds are tightened. In such cases, the cRN may be ignored entirely. Further conditions are identified as sufficient for satisfying the resource constraint, but allow for more flexibility in the timing constraints.

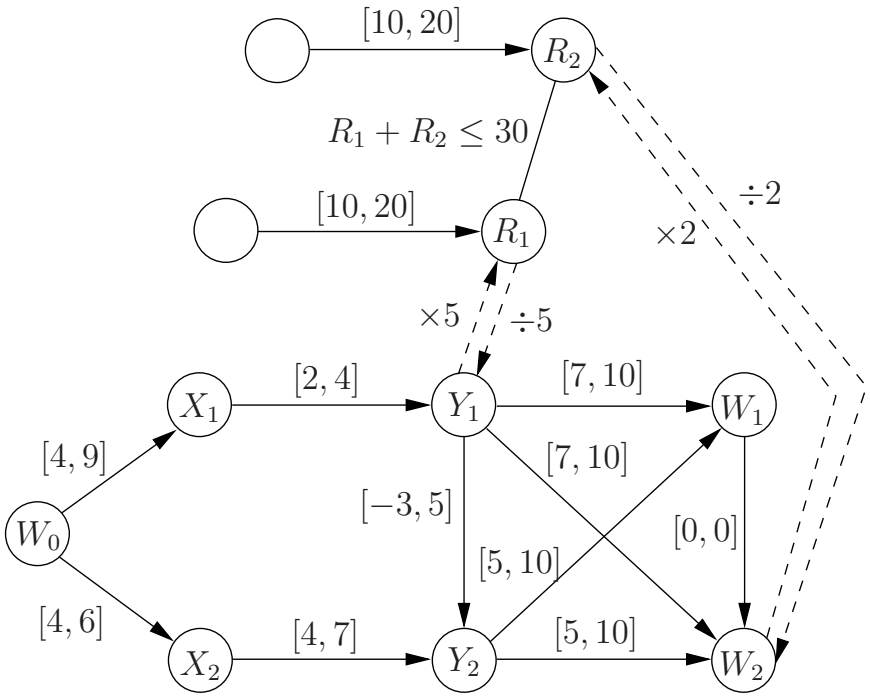


Figure 16: Example STN with a single resource cRN attached.

Two forms of interference are identified between the STN and cRN. First, a reduction in cRN upper bounds may delete values in the STN that are necessary to ensure dispatchability (cRN $\rightarrow$ STN interaction). Second, increasing the lower bound of an STN interval may require an increase in the lower bound of the corresponding resource interval in the cRN, thus violating the cRN dispatchability sufficient conditions (STN $\rightarrow$ cRN interaction). With these interactions in mind, a notion similar to the dominating edges in d-graphs is introduced, called a *buffer*, that allows for certain activities that use a resource to be disregarded.

[Wallace and Freuder, 2005] further provide a list of suggestions for solvers producing a schedule that is to be dispatched:

- Limit the length of the plan sent to the dispatcher so there are only a few instances of resource release.
- Arrange to have all of the capacity released at some point, effectively partitioning the network for dispatching.
- Minimize the number of buffers produced, as doing so will reduce the amount of interaction in the cRN.

In related work, [Laborie, 2003] presents an integrated method for planning and scheduling with resource constraints. Laborie introduces two constraints for propagating resource requirements: energy precedence constraints and balance constraints. The type of resource requirements considered may take on a discrete value, and events can either consume or produce units of resource. An activity may produce or consume a resource in a number of ways; at the start/end of an event, over the course of an event, etc. Similar to [Wallace and Freuder, 2005], the constraints are used primarily as a means of proving an inconsistency in the network, or tightening the bounds on simple temporal constraints. As such, the focus of [Laborie, 2003] is to use the constraints during the search for a valid schedule.

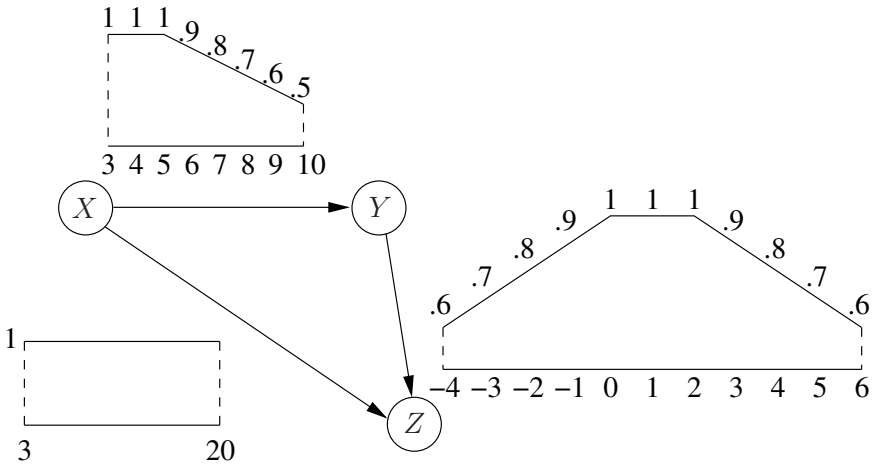


Figure 17: Example STPP with the preference profiles.

4.2 Preferences

Adding another layer of complexity to the problem, [Khatib et al., 2001] introduced preferences to STNs in a formalism called *Simple Temporal Problem with Preferences* (STPP). STPPs were later extended to include uncertainty with contingent links, leading to the *Simple Temporal Problem with Preferences and Uncertainty* (STPPU) [Rossi et al., 2006]. Rather than simply determining if a network is consistent or controllable, with preferences the goal is to find a dispatching strategy that maximizes some optimization function that depends on link durations. An optimal setting is one where there is no other choice of controllable time points that gives a higher global preference.

The preferences are given as a function that maps the duration of a temporal link to a particular value. In the case of an STPPU, a preference function on a contingent link can be viewed as the ideal preference situations that we would like to have, but cannot control. The functions that map a link duration to a preference value are restricted to a tractable class (fuzzy preferences and semi-convex functions), and the global preference value we want to optimize is the aggregate combination of every preference value on the links. Due to the choice of preference form, there is always a way to represent a hard constraint as a preference by assigning the preference value to be the max in the preference set. This property allows the formalism to deal only with preference links.

Example 7 (Example STPP)**.** In Figure 17, we show how a simple STN is augmented with preferences. The three temporal constraints are $[3, 10]_{XY}$, $[3, 20]_{XZ}$, and $[-4, 6]_{YZ}$. The overall preference quality of a schedule can be determined by the multiplication of each individual link preference, and we can see that XZ represents a simple temporal constraint – if it is violated then the overall preference will be 0. The preference profile for YZ can be interpreted as a preference that Z follows Y by no more than 2 units of time or they occur concurrently. We will accept something outside of these bounds (up to the hard upper and lower bounds), but the preference drops off linearly.

For strong and dynamic controllability, [Rossi et al., 2006] introduce a bounded parameter option for indicating the optimal preference will be satisfied if it is less than the bound, and if the optimal global preference is higher than the bound, then at least the given bound will be satisfied. It turns out that for the bounded case, introducing preferences to STPUs does *not* substantially increase the complexity of determin-

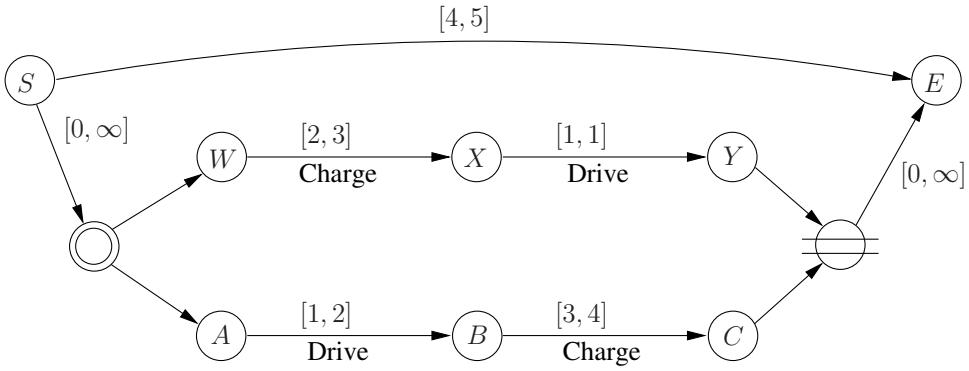


Figure 18: Example TPN with a choice node.

ing whether or not a network is controllable. For the weak controllability case, it is trivially determined: if the network without preferences is weakly controllable, then one can find the optimal setting of controllable time points at execution time by fixing the contingent link durations that are given, and using techniques for STPP consistency from [Khatib et al., 2001].

Despite the restriction on preference type, and the use of a bound for determining optimal control, [Rossi et al., 2006] note how positive the result is, since adding a considerable amount of expressive power through simple preferences on STPUs only increases the computational complexity by a linear amount in terms of the different preference levels.[7]

4.3 Temporally Flexible Plans

Beyond adding new restrictions to the edges in a network (such as the resources and preferences work), there is a branch of plan dispatchability research that focuses on generalizing the overall structure of the network. Nodes are no longer restricted to events alone, and edges are no longer restricted to temporal constraints.

The first and most widely adopted extension is the graphical counterpart to the *Reactive Model-based Programming Language* (RMPL), referred to as a *Temporal Plan Network* (TPN) [Kim et al., 2001]. The RMPL includes common programming constructs such as if-then-else, do-while loops, non-deterministic choice of program execution, etc. A direct translation is given to a network formalism (TPN) that is a generalization of an STN. The motivation behind this approach is that a user would specify the behaviour of the agent in RMPL, and the program is compiled to a TPN, and reasoned with by an online dispatcher.

Example 8 (TPN Example)**.** To give a general idea of the elements in a TPN, Figure 18 presents a simple network with a few of the advanced features present in a TPN. The double circle node represents a choice the agent must make, while the double line represents multiple choice threads merging together. Here, the agent can choose whether it should charge first, and then drive to its destination, or drive there first and then charge. Note that the durations depend on the order in which the agent decides to execute the activities.

Beyond the choice nodes, there are also constraints introduced for producing and consuming binary resources; represented as open links on nodes in the TPN that must be connected appropriately. Additional flexibility for the TPN was introduced in [Effinger et al., 2009] to handle *exceptions* that nature may

[7] It should be noted that the bound they use for checking controllability must be increased sequentially, as a binary search is not possible.

cause, and constructs to handle these exceptions in the TPN variant when they occur during dispatching. To add further interaction with nature, [Tsamardinos et al., 2003] introduces *Conditional Temporal Problems* (CTP) that allow for observation nodes, and events to be predicated on the existence of a particular set of observations.

Consistent dispatching of a TPN is the problem typically considered. A preliminary approach is presented in [Kim et al., 2001], and the dispatching method is expanded to use a leader election protocol in [Block et al., 2006] for a distributed computation of checking dispatchability. With the richer set of constructs (exceptions, etc), [Effinger et al., 2009] presents a method for dynamically executing the network in a form similar to dynamic controllability of STNUs. The difference here is that uncertainty is modelled through possible exceptions and not in the form of contingent links. Interestingly, the approach suggested is similar to the original approach for dynamic controllability – viewing the problem as a two-player game between the dispatcher and nature.

For the simpler forms of TPNs, [Conrad et al., 2009] (and later [Conrad and Williams, 2011]) draw a correspondence between the TPN and DTP formalisms. As we have described in Section 3.1.2, the approach adequately compiles the network into dispatchable form, and then dispatches the events while updating the labels appropriately. It should be noted that only a subset of the TPN capabilities were considered for this work (e.g., binary resource constraints are not allowed).

5 Relation to Plan Execution

In the area of automated planning, there is an analogue to online dispatching in the face of an uncertain world: *execution monitoring* (EM). While the field is quite extensive, there are some particular areas of EM that are closely related to the ideas presented in this survey. We mention some of them briefly here.

The first camp of EM research attempts to address uncertainty in the world by building robust plans and specialized dispatching methods to handle unexpected changes in the world during execution. Systems of this variety attempt to identify the conditions under which the plan remains satisfactory, and only attempt to repair a situation when these conditions become violated. Systems of this form include SIPE [Wilkins, 1985], Sage [Knoblock, 1995], Prodigy [Veloso et al., 1998], IxTeT-eXeC [Lemai and Ingrand, 2003], and more recently the work of Fritz [Fritz, 2009] and Muise [Muise et al., 2011]. All of the approaches assume that the world may change, but do not consider a specific model of uncertainty. Due to this philosophy of not modelling uncertainty explicitly, the approaches are primarily empirical in nature and do not provide theoretical guarantees on the dispatchability of the plan. This property also leads to the argument for a tighter integration between the planner and dispatcher, as described in [Knoblock, 1996], where (re)planning can and should be done in an online fashion when unexpected changes occur.

The other side of EM research adopts an approach more in line with the plan dispatchability we have seen above: uncertainty in the world is modelled and guarantees on execution are given. In [Musliner et al., 1995], a strategy is given to merge the planning and scheduling of tasks required of a real-time system. The system is able to provide both goal level and timing guarantees by producing a reactive loopy plan that avoids any state that might lead to unexpected behaviour (recall that uncertainty in the world is fully modelled). When the problem is oversubscribed, the system dynamically turns hard constraints into soft ones to satisfy as many of the requirements as possible. The system is also capable of taking on new goals as long as they can be effectively handled without violating a previously guaranteed goal or deadline.

Another approach that provides runtime guarantees is presented in [Chung and Williams, 2003]. Here, a reactive plan is constructed through exhaustive enumeration of valid world state and goal pairs. For any potential world state and goal, a strategy of reactive actions is pre-computed. This task is achieved through

a regression based approach that allows for decomposition of the world states based on independence of the potential goals. While they do not provide temporal guarantees, they do provide a goal-level guarantee for any potential change of the world model (as long as the world remains in a consistent state).

The focus for existing EM approaches is primarily to handle deviations of the world state, be it a heuristic method or a method with strong theoretical guarantees. There remains a large potential for overlap of EM techniques and the techniques we have surveyed.

6 Conclusion

In this paper, we have surveyed the field of plan dispatchability: how to effectively execute events in a schedule or plan while adhering to the temporal constraints placed between them. We have covered the various formalisms that incorporate uncertainty in the timing of events, and disjunctive constraints to allow non-convexity in the temporal requirements. With every formalism, two main problems are considered: how the plan can be checked for consistency (or controllability in the presence of uncertainty); and subsequently how to dispatch the plan effectively.

We have described how each variation is handled for checking consistency/controllability and dispatching the plan. We further presented the various extensions that have been considered, including the addition of resources, preferences, and flexible choice of event execution. There are connections between the planning literature for execution monitoring and the plan dispatchability techniques presented in this paper, as described in Section 5, but the opportunity remains for a far more integrated approach.

6.1 Future Directions

Aside from the integration with classical planning techniques, there are a number of areas that can be further developed in the plan dispatchability literature. Here, we present a selection of those identified while surveying the relevant literature:

1. *General DTNU*: Since the techniques for solving controllability of a DTNU are fairly new, there remains a possibility to further generalize the constraints permitted in a DTNU. As it stands, there are a number of assumptions placed on the DTNU that do not make it general: contingent links are assumed to take on a discrete value between the bounds; contingent links can only appear in a disjunctive constraint if all other disjuncts are contingent constraints on the same two events; etc. Removing these restrictions provides a more general model of DTNUs, and would require new techniques for determining the controllability of the network.

2. *Begin-Controllable Activities*: Only a handful of the approaches consider the possibility of a begin-controllable activity: the duration is controllable by the agent, but must be determined when the activity begins. This variation is a common requirement for plans/schedules that may be dispatched, and many of the approaches are not equipped to handle such information. The idea can be extended further to predicate the decision of an event's timing on the execution of other events (e.g., once both X and Y have been executed, the timing for Z must be chosen).

3. *Contingent Link Dependence*: Currently, the duration of any contingent link is assumed to be independent from the duration of any other link. Allowing the possibility for dependencies would allow for a far richer solution: e.g., the sum of two contingent links are guaranteed to be less than a certain bound. The notions of controllability also assume that we must be able to dispatch regardless of what

nature does. If a probability distribution was associated with the duration of a contingent link, then we could begin to ask questions of how to maximize the probability that the network is controllable.

4. *Mixed Controllability*: When considering a network with contingent links, it is assumed that the entire network should be controllable (either strongly, weakly, or dynamically). Lifting this assumption of uniform controllability would allow for a richer set of schedules to be produced – some areas of the network may need to be strongly controllable (as we may not be able to observe any of the events), while other areas may only need to be weakly or dynamically controllable. Such a generalization would require an entirely new set of techniques to decompose the network appropriately and mix the theoretical guarantees for each form of controllability considered.

References

J. Allen. Towards a general theory of action and time. *Artificial intelligence*, 23(2):123–154, 1984. 1

J. Bidot, T. Vidal, P. Laborie, and J. C. Beck. A theoretic and practical framework for scheduling in a stochastic environment. *Journal of Scheduling*, 12(3):315–344, 2009. ISSN 1094-6136. 2

S. A. Block, A. F. Wehowsky, and B. C. Williams. Robust execution on contingent, temporally flexible plans. In *International Conference on Automated Planning and Scheduling*, pages 802–808, 2006. 40

S. H. Chung and B. C. Williams. A decomposed symbolic approach to reactive planning. In *Proceedings of the Third International Workshop on Self-Adaptive Software*, 2003. 40

P. R. Conrad and B. C. Williams. Flexible Execution of Plans with Choice and Uncertainty. Technical report, 2011. 18, 21, 36, 40

P. R. Conrad, J. A. Shah, and B. C. Williams. Flexible execution of plans with choice. In *International Conference on Automated Planning and Scheduling*, pages 74–81, 2009. 18, 40

R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. In *Conference on the Principles of Knowledge Representation and Reasoning*, pages 83–93, 1989. 1, 4, 15

R. T. Effinger, B. C. Williams, G. Kelly, and M. Sheehy. Dynamic Controllability of Temporally-flexible Reactive Programs. In *International Conference on Automated Planning and Scheduling*, pages 122–129, 2009. 39, 40

C. Fritz. *Monitoring the Generation and Execution of Optimal Plans*. PhD thesis, University of Toronto, April 2009. 40

L. Hunsberger. Fixing the Semantics for Dynamic Controllability and Providing a More Practical Characterization of Dynamic Execution Strategies. In *Symposium on Temporal Representation and Reasoning (TIME)*, pages 155–162, 2009. 6, 11, 35

L. Hunsberger. A Fast Incremental Algorithm for Managing the Execution of Dynamically Controllable Temporal Networks. In *Symposium on Temporal Representation and Reasoning (TIME)*, pages 121–128, 2010. 12, 31, 33, 34

L. Khatib, P. Morris, R. Morris, and F. Rossi. Temporal constraint reasoning with preferences. In *International Joint Conference on Artificial Intelligence*, volume 17, pages 322–327, 2001. 38, 39

P. Kim, B. C. Williams, and M. Abramson. Executing reactive, model-based programs through graph-based temporal planning. In *International Joint Conference on Artificial Intelligence*, volume 17, pages 487–493, 2001. 39, 40

C. A. Knoblock. Planning, executing, sensing, and replanning for information gathering. In *International Joint Conference On Artificial Intelligence*, pages 1686–1693, 1995. 2, 40

C. A. Knoblock. Why Plan Generation and Plan Execution Are Inseparable. In *Proceedings of the AAAI Fall Symposium on Plan Execution*, 1996. 40

P. Laborie. Algorithms for propagating resource constraints in ai planning and scheduling: Existing approaches and new results. *Artificial Intelligence*, 143(2):151–188, 2003. 37

S. Lemai and F. Ingrand. Interleaving temporal planning and execution: IxTeT-eXeC. In *Proceedings of the ICAPS Workshop on Plan Execution*, 2003. 40

P. H. Morris. A structural characterization of temporal dynamic controllability. In *Conference on Principles and Practice of Constraint Programming*, pages 375–389. Springer, 2006. 31, 34, 35

P. H. Morris and N. Muscettola. Execution of Temporal Plans with Uncertainty. In *AAAI/IAAI*, pages 491–496, 2000. 29, 30

P. H. Morris and N. Muscettola. Temporal Dynamic Controllability Revisited. In *AAAI*, pages 1193–1198, 2005. 31

P. H. Morris, N. Muscettola, and T. Vidal. Dynamic control of plans with temporal uncertainty. In *International Joint Conference on Artificial Intelligence*, volume 17, pages 494–502, 2001. 30, 35, 36

C. Muise, S. A. McIlraith, and J. C. Beck. Monitoring the Execution of Partial-Order Plans via Regression. In *International Joint Conference On Artificial Intelligence*, 2011. 40

N. Muscettola, P. H. Morris, and I. Tsamardinos. Reformulating Temporal Plans for Efficient Execution. In *Conference on the Principles of Knowledge Representation and Reasoning*, pages 444–452, 1998. 14

D. J. Musliner, E. H. Durfee, and K. G. Shin. World modeling for the dynamic construction of real-time control plans. *Artificial Intelligence*, 74(1):83–127, Mar. 1995. ISSN 00043702. doi: 10.1016/0004-3702(94)00008-O. 40

B. Peintner, K. B. Venable, and N. Yorke-Smith. Strong Controllability of Disjunctive Temporal Problems with Uncertainty. In *Conference on Principles and Practice of Constraint Programming*, volume 4741 of *Lecture Notes in Computer Science*, pages 856–863. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-74969-1. doi: 10.1007/978-3-540-74970-7. 8, 25, 26, 27

L. Planken, M. De Weerdt, and R. van der Krogt. P3C: A new algorithm for the simple temporal problem. In *International Conference on Automated Planning and Scheduling*, pages 256–263, 2008. 15

F. Rossi, K. B. Venable, and N. Yorke-Smith. Uncertainty in soft temporal constraint problems: a general framework and controllability algorithms for the fuzzy case. *Journal of Artificial Intelligence Research*, 27(1):617–674, 2006. ISSN 1076-9757. 38, 39

S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003. 9

J. Shah, J. Stedl, B. C. Williams, and P. Robertson. A fast incremental algorithm for maintaining dispatchability of partially controllable plans. *International Conference on Automated Planning and Scheduling*, 2007. 22

J. A. Shah and B. C. Williams. Fast Dynamic Scheduling of Disjunctive Temporal Constraint Networks through Incremental Compilation. In *International Conference on Automated Planning and Scheduling*, pages 322–329, 2008. 22

J. A. Shah, P. R. Conrad, and B. C. Williams. Fast distributed multi-agent plan execution with dynamic task assignment and scheduling. In *International Conference on Automated Planning and Scheduling*, pages 289–298, 2009. 24

K. Stergiou and M. Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, 120(1):81–117, 2000. 5

I. Tsamardinos, M. E. Pollack, and P. Ganchev. Flexible dispatch of disjunctive plans. In *6th European Conference on Planning*, pages 417–422, 2001. 16, 21

I. Tsamardinos, T. Vidal, and M. E. Pollack. CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints*, pages 365–388, 2003. 21, 40

M. M. Veloso, M. E. Pollack, and M. T. Cox. Rationale-Based Monitoring for Planning in Dynamic Environments. In *Conference on Artificial Intelligence Planning Systems*, pages 171–180, 1998. 2, 40

K. B. Venable and N. Yorke-Smith. Disjunctive temporal planning with uncertainty. In *International Joint Conference on Artificial Intelligence*, pages 1721–1722, 2005. 8

K. B. Venable, M. Volpato, B. Peintner, and N. Yorke-Smith. Weak and Dynamic Controllability of Temporal Problems with Disjunctions and Uncertainty. In *Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS)*, 2010. 8, 28, 35, 36

T. Vidal. A unified dynamic approach for dealing with temporal uncertainty and conditional planning. In *Conference on Artificial Intelligence Planning Systems*, pages 395–402, 2000. 29

T. Vidal and H. Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental & Theoretical Artificial Intelligence*, 11(1):23–45, 1999. ISSN 0952-813X. doi: 10.1080/095281399146607. 6, 9, 10, 11, 24, 27, 29

T. Vidal and M. Ghallab. Dealing with Uncertain Durations In Temporal Constraint Networks dedicated to Planning. In *European Conference on Artificial Intelligence*, pages 48–54, 1996. 24

R. J. Wallace and E. C. Freuder. Supporting dispatchability in schedules with consumable resources. *Journal of Scheduling*, 8(1):7–23, 2005. ISSN 1094-6136. doi: 10.1007/s10951-005-5313-7. 36, 37

D. Wilkins. Recovering from execution errors in sipe. *Computational Intelligence*, 1(1):33–45, 1985. 2, 40

L. Xu and B. Y. Choueiry. A new efficient algorithm for solving the simple temporal problem. In *Proceedings of the 10th International Symposium on Temporal Representation and Reasoning and Fourth International Conference on Temporal Logic*, 2003. 14

H. L. S. Younes and R. G. Simmons. Vhpop: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research*, 20:405–430, 2003. 12

A Glossary

- *AllMax projection* (pg. 32): The boundary projection that replaces every contingent link with a simple temporal constraint that forces the duration to be the upper bound.
- *boundary projection* (pg. 29): A projection where every contingent link is set to equal either the upper or lower bound.
- *choice variables* (pg. 18): A variable that represents the choice of which disjunct to satisfy from a disjunctive constraint.
- *chordal graph* (pg. 15): A graph where every cycle of length greater than 3 has an edge that cuts the cycle into two smaller ones.
- *complete situation* (pg. 7): A chosen duration for every contingent link in a network with uncertainty.
- *component STN* (pg. 5): The STN that results in selecting a simple temporal constraint from every disjunction in a DTN or TCSN.
- *component STP* (pg. 5): The STP of a component STN.
- *Conditional Temporal Problems* (pg. 40): A generalization of STNs that allow observation nodes, and events to be predicated on the existence of a set of observations.
- *consistent* (pg. 2): Property of network that holds when it can be effectively executed.
- *consumable resource network* (pg. 36): A network used to represent the resource constraints in an STN with resources.
- *contingent constraint* (pg. 6): A temporal constraint between two time points whose duration is chosen by nature (i.e., out of the agent's control). It is assumed that the lower bound is 0 or higher.
- *contingent-labelled distance graph* (pg. 31): A labelled multi-graph with events as nodes and either weights, upper-case labels, or lower-case labels on the edges.
- *control sequence* (pg. 7): In an STNU, a set of assignments to executable events.
- *CTP*: See Conditional Temporal Problems.
- *Disjunctive Temporal Network* (pg. 5): An STN that allows for disjunctive temporal constraints where the disjuncts may be arbitrary simple temporal constraints.
- *Disjunctive Temporal Network with Uncertainty* (pg. 8): A DTN that allows for disjuncts to be contingent constraints as well as free constraints.
- *Disjunctive Temporal Problem* (pg. 5): The problem of determining if a DTN is consistent.
- *Disjunctive Temporal Problem with Uncertainty* (pg. 9): The problem of determining if a DTPU is controllable (either weakly, strongly, or dynamically).
- *dispatchable form* (pg. 12): A network that can be effectively dispatched in an online fashion, usually in low polynomial time.

- *domination function* (pg. 19): A binary function that induces a total ordering on the elements in its domain.
- *DTN*: See Disjunctive Temporal Network.
- *DTN time window* (pg. 16): Union of time windows from each component STP in a DTN.
- *DTNU*: See Disjunctive Temporal Network with Uncertainty.
- *DTP*: See Disjunctive Temporal Problem.
- *DTPU*: See Disjunctive Temporal Problem with Uncertainty.
- *dynamic back-propagation rules* (pg. 22): Propagation rules for an STN that makes the network dispatchable after a constraint has been tightened or added.
- *dynamic controllability* (pg. 10): A property of networks with uncertainty that allows an agent to make choices dynamically during execution based only on the knowledge of when observed events in the past have happened.
- *dynamic execution strategy* (pg. 11): An execution strategy that behaves in the same manner at any situation that has the same pre-history.
- *environment* (pg. 18): A (possibly partial) setting to the choice variables in a DTN.
- *executable events* (pg. 6): Any event that is not at the end of a contingent link.
- *execution strategy* (pg. 11): A mapping from complete situations to schedules.
- *extension* (pg. 32): A path in a contingent-labelled distance graph that follows a lower-case edge.
- *free constraints* (pg. 7): Temporal constraints that are not contingent.
- *labelled distance graph* (pg. 18): The distance graph that is associated with a labelled STN.
- *labelled STN* (pg. 18): An STN that has labelled value sets on the edges.
- *labelled value set* (pg. 19): A set of pairs that consist of a value and an environment that is kept minimal based on some measure of minimality that involves a dominance function.
- *moat edge* (pg. 33): The final edge in the path of an extension.
- *observed events* (pg. 6): An event that is at the end of a contingent link.
- *pre-history* (pg. 11): All of the durations to contingent links in the past, conditioned on a time point.
- *projection* (pg. 7): The STN that is formed by replacing every contingent link with a simple temporal constraint based on its setting in a complete situation.
- *pseudo-controllable* (pg. 30): An STNU that, when treated as an STN and checked for consistency, doesn't tighten any contingent link.
- *Reactive Model-based Programming Language* (pg. 39): A simple language for modelling the behaviour of an agent.

- *real-time execution decision* (pg. 35): A decision that an agent may make while dispatching a network that is dynamically controllable.
- *reduced distance* (pg. 32): In a contingent-labelled distance graph, the sum of weights on edges in a path (ignoring the distinction between upper-case, lower-case, and regular edges).
- *reducible path* (pg. 32): A path in a contingent-labelled distance graph that contains two edges that can be reduced by the edge reduction rules.
- *reduction rules*: Inference rules for adding new edges to a contingent-labelled distance graph.
- *RMPL*: See Reactive Model-based Programming Language.
- *RTED*: See real-time execution decision.
- *safe network* (pg. 30): An STNU that is structured so bounds propagated to an observed event always go through the contingent link.
- *schedule* (pg. 3): A function that maps events to times the event should be executed.
- *semi-executable* (pg. 8): Disjunctive constraints in a DTNU that mix both free and contingent constraints.
- *semi-reducible* (pg. 32): A path in a contingent-labelled distance graph that can be reduced through repeated application of reduction rules to a path that contains no lower-case edge.
- *Simple-Nature* (pg. 27): A DTNU that contains no disjunctive constraint made up of contingent constraints.
- *Simple-Nature-Dependent* (pg. 26): A DTNU where every disjunctive constraint is either made up of contingent constraints, or over only executable events.
- *simple temporal constraint* (pg. 3): A lower bound and upper bound between the time of two events. The lower bound is assumed to be no greater than the upper bound, and both can be any real number, including positive and negative infinity.
- *Simple Temporal Network* (pg. 2): A network that consists of events as nodes, and simple temporal constraints between the events as edges.
- *Simple Temporal Network with Uncertainty* (pg. 6): A network that consists of events as nodes, and simple temporal constraints or contingent constraints between the events as edges.
- *Simple Temporal Problem* (pg. 2): The problem of determining if an STN is consistent.
- *Simple Temporal Problem with Preferences* (pg. 38): The problem of optimizing the dispatch of an STN where preferences are associated to activity durations.
- *Simple Temporal Problem with Preferences and Uncertainty* (pg. 38): The problem of determining controllability of a STPP where contingent links are added.
- *Simple Temporal Problem with Uncertainty* (pg. 8): The problem of determining if an STNU is controllable (either weakly, strongly, or dynamically).

- *simplical* (pg. 15): A node in a graph whose neighbours induce a clique.
- *simplical elimination ordering* (pg. 15): An ordering of vertices such that every removed vertex is simplical in the graph that remains.
- *situation* (pg. 7): See complete situation.
- *space of complete situations* (pg. 7): In an STNU, the complete set of all possible choices nature can assign to the contingent links (i.e., all of the complete situations).
- *STN*: See Simple Temporal Network.
- *STNU*: See Simple Temporal Network with Uncertainty.
- *STP*: See Simple Temporal Problem.
- *STPP*: See Simple Temporal Problem with Preferences.
- *STPPU*: See Simple Temporal Problem with Preferences and Uncertainty.
- *STPU*: See Simple Temporal Problem with Uncertainty.
- *strongly controllable* (pg. 9): A property of networks with uncertainty that allows an agent to give a single control sequence regardless of the environment's choices of contingent link duration.
- *Temporal Constraint Satisfaction Network* (pg. 4): An STN that allows for disjunctive temporal constraints where the disjuncts may be simple temporal constraints between the same two events.
- *Temporal Constraint Satisfaction Problem with Uncertainty* (pg. 8): A TCSN that allows for disjuncts to be contingent constraints as well as free constraints.
- *Temporal Constraint Satisfaction Problem* (pg. 4): The problem of determining if a TCSN is consistent.
- *Temporal Plan Network* (pg. 39): A generalization of STNs that allow for choice nodes, binary resource constraints, etc.
- *TCSN*: See Temporal Constraint Satisfaction Network.
- *TCSNU*: See Temporal Constraint Satisfaction Problem with Uncertainty.
- *TCSP*: See Temporal Constraint Satisfaction Problem.
- *tight edge* (pg. 30): An edge (X, Y) in the distance graph of an STN such that no shorter path exists between X and Y.
- *Totally Simple* (pg. 26): A DTNU that has no contingent constraints in a disjunctive constraint.
- *TPN*: See Temporal Plan Network.
- *wait constraint* (pg. 30): A temporal constraint between three events X, Y, and Z that indicates Y must wait until either t time has passed after event X is executed, or event Z is executed.
- *weakly controllable* (pg. 10): A property of networks with uncertainty that allows an agent to give a control sequence that depends on a complete situation.