Optimising Flexibility of Temporal Problems with Uncertainty

Jing Cui

Australian National University

A thesis submitted for the degree of
DOCTOR OF PHILOSOPHY
The Australian National University

July 2018

Except where otherwise indicated, this thesis is my own original work.

Jing Cui
7 July 2018

To my mother, Lingling Han.

Acknowledgments

The work presented in the thesis would not have been possible without the support of a number of individuals and organisations, and they are gratefully acknowledged below.

First of all, I want to extend my sincerest gratitude to my supervisor, **Dr. Patrik Haslum**. He is a knowledgeable and experienced expert in the field of artificial intelligence. He patiently and wisely supervised my research and was a constant source of ideas and encouragement. He is always willing to help me improve my knowledge and broaden my view. He supported my attendances for ICAPS 2014, 2015 and 2017, the NICTA Optimisation Summer School 2014 and the Recognition Robotics Summer School 2017. They were great opportunities to present and communicate my research with top scientists and students, during different stages of my study.

I would like to thank my advisors **Prof. Sylvie Thiébaux** and **Dr. Hassan Hajiazi**. They always provided valuable feedback and helpful suggestions that help me summarise the high-level view of my research topic and solve the difficulties in the thesis. I thank all staffs and students in NICTA, which is called DATA61 now, for their generous assistance, inspiring discussions and wonderful playing time, i.e. the board-game nights and the foosball matches. It was my great pleasure to study in such a friendly and relaxing atmosphere.

During the study, I was fortunate to be able to work with **Dr. Peng Yu** and **Cheng (Simon) Fang** from **Prof. Brian C. Williams**'s group. Thanks for their brilliant views and generous assistance, we collaborated successfully and got our work recognised in the research community. Special thanks to the people in MERS of MIT, who provided friendly help and valuable discussion during my visits in 2014 and 2017.

I would like to extend my gratitude to **Prof. Weiming Zhang** and **Prof. Cheng Zhu**. Thanks for their support I got the opportunity to study in Australia. I also want to thank Dr. Jiangfeng Luo who introduced me to the field of temporal planning and scheduling six years ago, so that I could enjoy an exciting and promising research field during my PhD.

Thanks must go to the Chinese Scholarship Council, the Australian National University, DATA61 (NICTA) for providing the PhD scholarship and tuition fee, and a 2017 ANU Vice Chancellor's Travel Grant and the ICAPS Doctoral Consortium for supporting my conference attendances, respectively.

I would like to give my gratitude to my dear parents for their continuous support in general. Special thanks to my boyfriend, Yan Zhao for his encouragement and understanding. Finally, I would like to thank all my friends in Australia, who provided me with a homely environment overseas.

Abstract

Temporal networks have been applied in many autonomous systems. In real situations, we cannot ignore the uncertain factors when using those autonomous systems. Achieving robust schedules and temporal plans by optimising flexibility to tackle the uncertainty is the motivation of the thesis.

This thesis focuses on the optimisation problems of temporal networks with uncertainty and controllable options in the field of Artificial Intelligence Planning and Scheduling. The goal of this thesis is to construct flexibility and robustness metrics for temporal networks under the constraints of different levels of controllability. Furthermore, optimising flexibility for temporal plans and schedules to achieve robust solutions with flexible executions.

When solving temporal problems with uncertainty, postponing decisions according to the observations of uncertain events enables flexible strategies as the solutions instead of fixed schedules or plans. Among the three levels of controllability of the Simple Temporal Problem with Uncertainty (STPU), a problem is dynamically controllable if there is a successful dynamic strategy such that every decision in it is made according to the observations of past events.

In the thesis, we make the following contributions. (1) We introduce an optimisation model for STPU based on the existing dynamic controllability checking algorithms. Some flexibility and robustness measures are introduced based on the model. (2) We extend the definition and verification algorithm of dynamic controllability to temporal problems with controllable discrete variables and uncertainty, which is called Controllable Conditional Temporal Problems with Uncertainty (CCTPU). An entirely dynamically controllable strategy of CCTPU consists of both temporal scheduling and variable assignments being dynamically decided, which maximize the flexibility of the execution. (3) We introduce optimisation models of CCTPU under fully dynamic controllability. The optimisation models aim to answer the questions how flexible, robust or controllable a schedule or temporal plan is. The experiments show that making decisions dynamically can achieve better objective values than doing statically.

The thesis also contributes to the field of AI planning and scheduling by introducing robustness metrics of temporal networks, proposing an envelope-based algorithm that can check dynamic controllability of temporal networks with uncertainty and controllable discrete decisions, evaluating improvements from making decisions strongly controllable to temporally dynamically controllable and fully dynamically controllable and comparing the runtime of different implementations to present the scalability of dynamically controllable strategies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Autonomous artificial intelligence (AI) systems are becoming more and more popular these days. They have been applied not only in projects that cost a fortune such as aerospace engineering, autonomous underwater vehicles or other robotics systems but also in areas closer to daily life, for instance, self-driving cars, smart home and other applications. How autonomous systems execute robustly and flexibly in the real world is a core issue of artificial intelligence. The uncertainty of the actual environment is one of the most common factors reducing the performance or even breaking the systems. To tackle the uncertainty, a robust autonomous system has to either react according to the observation of unforeseen circumstances or execute flexibly enough to absorb the unexpected events. This goal can be achieved by estimating the uncertainty, generating alternatives based on the estimations and adjusting the execution according to the exact situations. Therefore, when building an autonomous system, we want to make it robust and flexible to deal with the uncertain environment. In this thesis, we focus on the field of robustness and flexibility of autonomous systems.

Artificial intelligence planning and scheduling are one of the leading techniques to build an autonomous system. They play a role of "thinking" in an autonomous system and coordinate with other techniques that play roles of "sensing" and "acting". In AI planning and scheduling, the solutions are either a plan, which is a sequence of actions that can transfer the environment from an initial state to the goal state, or a schedule, which decides how to allocate resources and when to execute each event to satisfy all constraints. The robustness of a temporal plan or schedule is a critical issue when applying it to the real world. In the research of temporal planning and scheduling, flexible solutions such as Partial Order Plans [Weld, 1994] and Partial Order Schedules [Policella et al., 2007] instead of fixed schedules and plans enable dynamic executions [Muise, 2014] to deal with the uncertain situations. A flexible solution of a planning and scheduling system contains several alternative solutions just in case the original one does not work. The choice among those alternatives is made during the execution, when unexpected situations happen. Postponing decisions enables adjusting the execution based on observations of the situations that have occurred.

Using temporal reasoning models to represent temporal problems is a useful technique in AI planning and scheduling. Different temporal reasoning models made

of timepoints, temporal constraints and other elements are widely used in popular planning and scheduling systems. One advantage of using temporal models is that it simplifies the problem to a satisfaction problem, which helps to analyse the robustness while the uncertain environment can be abstracted in the constraint model. Adding uncertainty into temporal reasoning models, such as Simple Temporal Problems [Dechter et al., 1991] enables us to study uncertainty directly. Vidal and Fargier [1999] introduced Simple Temporal Problems with Uncertainty (STPU) that assume intervals can represent the durations of uncontrollable events. Before execution, the uncontrollable durations are unknown. During execution, the exact durations may be any value within their intervals. When the durations of the uncertain events are not observable, it requires a universal solution that can deal with any uncertain situations. If such a solution exists, the problem is strongly controllable (SC). Additionally, if the uncertain durations are observable after they have finished, it enables a dynamic strategy to adjust to different past situations. If such a dynamic strategy exists, the problem is dynamically controllable (DC). The dynamically controllable strategy of STPU postpones decisions on time points to allow more flexibility to deal with temporal uncertainty rather than a fixed schedule.

Additionally, different flexibility and robustness metrics have been introduced to measure the quality of plans and schedules. Implementing robustness metrics in planning and scheduling systems may help to produce robust solutions. Robustness models built on temporal reasoning models reflect the quality of temporal plans and schedules.

This thesis aims to improve flexibility and robustness of schedules and temporal plans by considering the robustness of flexible solutions of planning and scheduling systems.

1.1 Problem Statement

Temporal reasoning is a fundamental technique in scheduling and planning. Dechter et al. [1991] introduced the Simple Temporal Networks (STN) that consists of timepoints and temporal constraints between pairs of timepoints. The timepoints can be regarded as variables, and the temporal constraints are lower and upper bounds from one timepoint to the other. An assignment to all timepoints that can satisfy every constraint is a feasible solution. If a problem has a feasible solution, the network representing the problem is consistent.

However, STN cannot represent the uncertainty in the real world. Different kinds of uncertainty exist in real scheduling problems, e.g., the train scheduling with possible delays caused by natural or human factors, the personal travel planning with fluctuating public transportation timetables and manufacture process scheduling with possibly not working machines.

1.1.1 Optimising Temporal Problem with Uncertainty under Controllability Constraints

STN with Uncertainty (STNU) or Simple Temporal Problem with Uncertainty (STPU) was introduced by Vidal and Fargier [1999] as an extension of STN. STPU has uncontrollable timepoints and temporal constraints. The durations of uncontrollable links are decided by the environment, not the scheduler or the agent executing the problem, but they can be observed when completed. In other words, the durations may be any value within their bounds. Because of the uncertainty, Vidal and Fargier [1999] discussed the three levels of controllability: strong controllability, dynamic controllability and weak controllability. Strong controllability is the most strict one. An STPU is strongly controllable if there is a consistent solution that can deal with all uncertain situations. An STPU is weakly controllable if, for every uncertain situation, there is a consistent solution available. Between those two, an STPU is dynamically controllable, if there will always be successive decisions based on the past observations, no matter what happens in the uncertain environment. Detailed definitions of the controllability are given in the later chapters.

The weak controllability requires knowing all durations of the uncontrollable events before execution, which makes it not very applicable. The strong controllability and dynamic controllability are both useful, but the strong controllability is so strict that it may be too hard to achieve in many situations. In the thesis, we use the dynamic controllability as a primary property and implement the strong controllability as a comparing factor.

Dynamically controllable simple temporal problems with uncertainty (STPU) are widely used to represent temporal plans and schedules that can execute flexibly based on the observability of the uncertain events. While the problem of testing an STPU for dynamic controllability is well studied, many use cases – for example, relaxing over-constrained temporal networks Beaumont et al. [2001, 2004] and analysing robustness of temporal plans and schedules Jorge Leon et al. [1994]; Wu et al. [1999]; Goren and Sabuncuoglu [2008] – require optimising a function over STPU time bounds subject to the constraint that the network is dynamically controllable.

We present a disjunctive linear constraint model of dynamic controllability, show how it can be used to formulate a range of applications and compare a mixed-integer, a non-linear programming, and a conflict-directed search solver on the resulting optimisation problems. Our model also provides the first solution to the problem of optimisation over a probabilistic STN subject to dynamic controllability and chance constraints.

1.1.2 Checking Dynamic Controllability for Temporal Problems with Uncertainty and Choices

Besides focusing on temporal problems with uncertainty, we extend the work by adding discrete variables as choices. Although it is far simpler than scheduling and

planning problems, the discrete variables may contribute to representing other constraints such as resource constraints or choices between events/actions. The discrete variables whose assignments can be attached to activate or deactivate constraints are called "controllable conditions" because they are chosen by the agent and they are the conditions to invoke constraints.

From Conditional Temporal Problem (CTP) [Tsamardinos et al., 2003], which attaches a label of predicates to a node to represents the situations in which the node will be executed, Conditional STNU [Hunsberger et al., 2012] and Controllable Conditional Temporal Problem with Uncertainty (CCTPU) [Yu et al., 2014] are introduced by adding uncertainty. Conditional STNU extends to uncertain conditions that are not controllable but observable after a specific observation timepoint. After the observation, specific links attached to observed conditions are activated. However, in this thesis, we do not discuss the uncertain conditions but focus on the controllable conditions that can represent the choices.

By considering choices in temporal problems, the decisions of both choices and temporal points can be made dynamically or strongly controllable. Making both decisions strongly controllable means there has to be a strongly controllable STPU with fixed options of the choices that activate this STPU. Making only temporal decisions dynamically means the STPU activated by fixed options of the choices has to be dynamically controllable. Making both decisions dynamically means there is a fully dynamic strategy that can make successive temporal decisions and options based on past observations. The more decisions we postponed to make them dynamically, the more flexible and robust the solution may be.

We extend the concept of dynamic controllability to the Controllable Conditional Temporal Problem with Uncertainty (CCTPU), which extends the STPU by conditioning temporal constraints on the assignment of controllable discrete variables. We define dynamic controllability of a CCTPU as the existence of a strategy that decides on both the values of discrete choice variables and the scheduling of controllable time points dynamically. This contrasts with the previous work on CCTPU, which considered a static assignment of choice variables and dynamic decisions over time points only. We propose an algorithm to find such an entirely dynamic strategy. The algorithm computes the "envelope" of outcomes of temporal uncertainty in which a particular assignment of discrete variables is feasible, and aggregates these envelopes over all choices. When an aggregated envelope covers all uncertain situations of the CCTPU, the problem is dynamically controllable. The experiments on an existing set of CCTPU benchmarks show that there are cases in which making both discrete variable and temporal decisions dynamically it is feasible to satisfy the problem constraints while assigning the discrete variables statically it is not.

1.1.3 Optimising Temporal Problems with Uncertainty and Choices under Controllability Constraints

In addition to answering if a CCTPU is dynamically controllable or not, we want to answer how robust, flexible or controllable a CCTPU is. To answer this question, we

introduce optimisation models for the CCTPU. In optimisation models, the variables represent the bounds of links and the options of the choices, the constraints represent strong controllability, temporally dynamic controllability and entirely dynamic controllability and the objective functions are features related to robustness, flexibility or other preferences. By solving the models, we can answer what the best value the problems can achieve when their solutions are in different levels of controllability.

1.2 Contribution

The main contributions of this thesis are the follows.

- We introduce a disjunctive linear constraint model for STPU under dynamic controllability, in order to optimise the flexibility and robustness of temporal problems with uncertainty. The disjunctive linear model can be encoded into a Mixed Integer Programming (MIP) problem or a non-linear programming problem that can be solved by existing solvers.
- We introduce several worst-case and average-case robustness metrics base on the optimisation model and compare them with existing robustness and flexibility measures to compare the differences and figure out the exact features the metrics measure.
- We extend the dynamic controllability to temporal problems with uncertainty and controllable options of choices, a CCTPU. The fully dynamically controllable strategy of a CCTPU makes temporal scheduling and controllable options dynamically controllable. In order to represent such a strategy, we make assumptions on the decision timepoints of the controllable choices and introduce an envelope-based algorithm to verify the existence of such a strategy.
- We propose optimisation models of CCTPU under different levels of controllability with fixed and dynamic options of the choices so that the optimisation model can answer how flexible or controllable a CCTPU is.
- Among constraint models of strong controllability, temporally dynamic controllability with fixed options of choices and fully dynamic controllability, we evaluate the reductions of relaxation cost to present how much improvement can be achieved by making decisions more and more flexible. At the same time, we also compare the runtime to solve these constraint models so that we can briefly show the scalability of solving optimisation problems in different levels of controllability.

1.3 Thesis Outline

In Chapter 2, we illustrate the background of the thesis. In Chapter 3, we describe our work on the problem of optimising STPU, which has been published in a conference paper [Cui et al., 2015] and involved in a journal article [Yu et al., 2017]. In

Chapter 4, we present our contribution to dynamic controllability checking of STPU with dynamic options, which is an extension of another conference paper [Cui and Haslum, 2017]. In Chapter 5, we describe the optimisation model of CCTPU. Finally, we conclude in Chapter 6, discuss other related works and discuss possible future research directions.

d

Chapter 2

Background

This chapter presents the necessary background and notations required for the technical chapters of the thesis. We first describe temporal reasoning models representing temporal problems. Then we describe the three levels of controllability of Simple Temporal Problems with Uncertainty, among which dynamic and strong controllability are used in the following chapters. We finish with an overview of existing robustness and flexibility measures of temporal networks with which we use to compare in the thesis.

2.1 Temporal Reasoning Models

In this section, we briefly review the developments of the temporal reasoning models used in the thesis.

Temporal reasoning models are used to represent temporal logic and inspect implicit temporal factors of real problems. A temporal problem usually consists of time points, temporal constraints and other features such as conditions, uncertainties or preferences. A solution to a temporal problem contains the assignments of the time points that satisfy all temporal constraints and respect to the other features of the problems. Schwalb and Vila [1998] claimed that the main tasks of temporal reasoning are (1) deciding consistency of the temporal networks and (2) finding solutions to temporal problems.

Solving a temporal problem can be viewed as solving a Constraint Satisfaction Problem (CSP), which consists of decision variables, domains of the variables and constraints among the variables. In terms of the temporal problems, the variables of the CSP are either time points or temporal intervals. Time points represent the start and end of events. Time intervals represent time periods during which the events happen. Temporal constraints can be regarded as constraints among variables. Qualitative temporal reasoning models such as interval algebra [Allen, 1983] and point algebra [Vilain et al., 1986] have the ability to describe the qualitative relations between pairs of events, such as before, after or overall. Quantitative reasoning models can model the durations of events. A quantitative temporal constraint between timepoints x_i and x_j such as $l_{ij} \leq x_j - x_i \leq u_{ij}$ describes that the duration from x_i to x_j is within $[l_{ij}, u_{ij}]$.

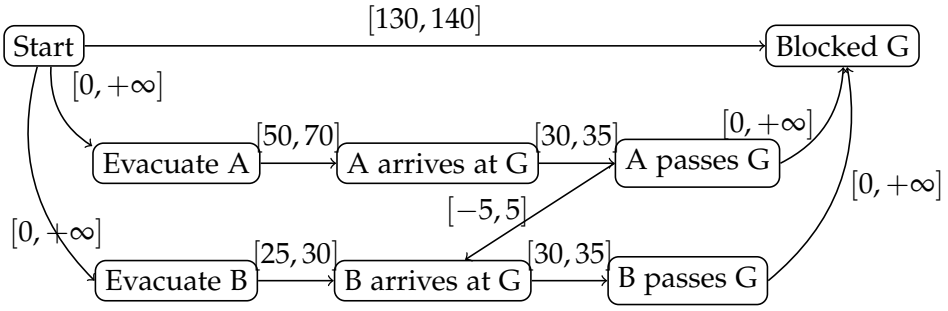


Figure 2.1: An STN example

Dechter et al. [1991] introduced the Temporal Constraint Satisfaction Problem (TCSP) made of a set of timepoints as variables $\{x_1, x_2, ..., x_n\}$, the continuous domains for variables and a set of constraints. Each constraint is a disjunction of intervals for a pair of variables $(l_1 \leq x_j - x_i \leq u_1) \vee ... \vee (l_k \leq x_j - x_i \leq u_k)$.

Dechter et al. [1991] also introduced a special class of TCSP, the Simple Temporal Networks (STP), in which the constraints are single intervals (without disjunctions). The STN is a common model used for temporal reasoning.

Definition 2.1. A **Simple Temporal Network (STN)** is a tuple, $< X, E >$, where:

- $X = x_1, x_2, ..., x_n$ is the set of time points,
- $E = e_1, e_2, ..., e_m$ is the set of temporal constraints, each constraint $l_{ij} \leq x_j - x_i \leq u_{ij}$ describes the duration from x_i to x_j is within $[l_{ij}, u_{ij}]$. A **Simple Temporal Problem (STP)** is the problem of solving a given STN.

For example in the evacuation planning problem [Even et al., 2014], before roads are blocked by the flood, a successful plan should evacuate all people. The evacuation plan consists of assignments of a path to each region and the time to execute each evacuating action. After assigning a specific route to each region, the STN can represent the temporal network of an evacuation plan. Timepoints represent when evacuating from each region and when each region arrives key positions can be. The durations to drive from one position to another position and the time until the flood comes to each key positions can be modeled as temporal constraints. Figure 2.1 shows an STN example of the evacuation planning: it means that the traffic flows evacuating from regions A and B have to pass G before the flood blocks G and those from region B will follow those from region A to pass G.

An evacuation plan of the STN shown in the example can be: (1) Region A evacuates at time $t = 0$, it arrives G at $t = 50$ and it passes G at $t = 80$; (2) Region B evacuates at $t = 55$, it arrives G at $t = 80$, which is immediately after Region A passing G, and B passes G at $t = 110$; (3) Region A and B both pass G before the flood blocks G ($t = 130$).

Based on the STN, different temporal reasoning models have been introduced according to application requirements, as shown in Figure 2.2. The Simple Temporal

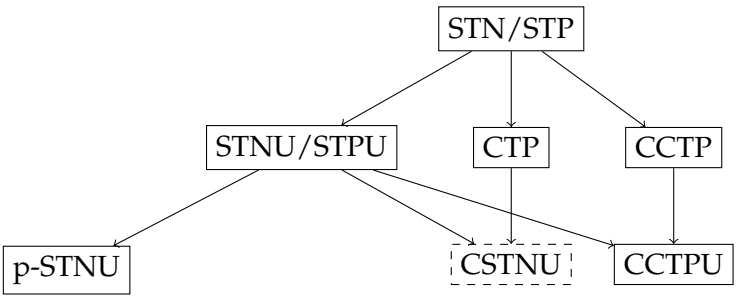


Figure 2.2: Temporal Reasoning Models

Network with Uncertainty (STNU) or Simple Temporal Problem with Uncertainty (STPU) [Vidal and Fargier, 1999] extends the STN by adding uncertain links. The Conditional Temporal Problem (CTP) [Tsamardinos et al., 2003] extends the STN by adding uncertain conditions that can activate/deactivate timepoints in the STN. Controllable Conditional Temporal Problem (CCTP) [Yu and Williams, 2013] extends the STP by adding conditions into controllable decisions among options and attaching the conditions to the links. The Controllable Conditional Temporal Problem with Uncertainty (CCTPU) [Yu et al., 2014] combines STPU and CCTP, which is a temporal problem with uncertainties and decisions of controllable choices. CCTPU is different from another temporal model – the Conditional Simple Temporal Network with Uncertainty (CSTNU) [Hunsberger et al., 2012] – which also combines the STPU and the CTP but considers uncontrollable conditions. In this thesis, we also use probability-STNU (p-STNU) [Tsamardinos, 2002; Fang et al., 2014] that represents uncertainty by probability distributions instead of lower and upper bounds.

Other temporal reasoning models and related research will be discussed in Chapter 6.

2.1.1 Simple Temporal Network with Uncertainty (STNU/STPU)

An STNU/STPU is a constraint satisfaction problem over real-valued time point variables, with constraints that are (upper and lower) bounds on the differences between pairs of variables. However, some time points are *uncontrollable*, meaning that in any execution of the schedule or plan, their values will be chosen non-deterministically (by the environment) within the given bounds, while the values of remaining time point variables are selected by the executing agent, subject to the constraints. The STPU is widely used to model temporal problems with duration uncertainty. It extends the STN by adding uncertain timepoints and constraints.

Both STNU and STPU are widely used by different authors [Vidal and Fargier, 1999; Morris et al., 2001; Hunsberger, 2009]. We use STPU, which was the first name given by Vidal and Fargier [1999], in the thesis.

Definition 2.2. An **STPU** [Vidal and Fargier, 1999] is a tuple, $< V, E >$.

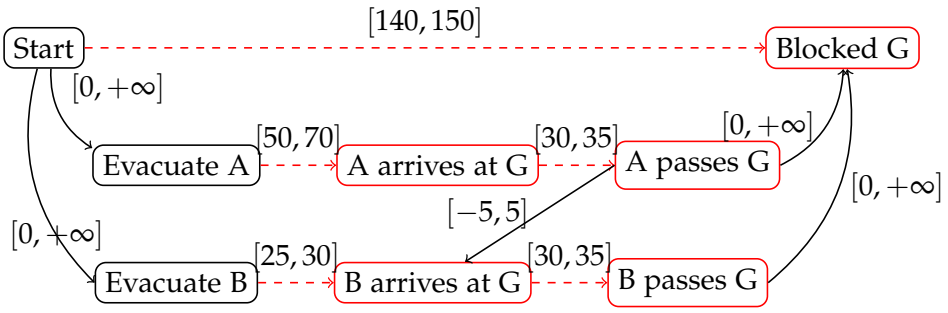


Figure 2.3: An STPU example

- V is a set of nodes $V = V_E \cup V_U$, representing executable (V_E) and uncontrollable (V_U) time points,
- $E = EC \cup EU$ is a set of links, called *requirement* and *contingent* links, representing controllable and uncontrollable links, respectively. Each link e_{ij} has a lower bound L_{ij} and upper bound U_{ij}, representing the constraints $L_{ij} \leq t_j - t_i \leq U_{ij}$.

Each uncontrollable time point has exactly one incoming contingent link, whose lower bound is non-negative. In other words, executable time points correspond to choices of the agent, contingent links represent uncontrollable durations, and the uncontrollable time points are when the agent finds out what duration the environment has chosen. Requirement links may connect any pair of time points.

Since contingent links in an STPU are not controllable, their actual durations do not belong to the solution decided by the scheduler. The following definitions are given to define a solution to an STPU [Vidal and Fargier, 1999]. For example in Figure 2.3, the red nodes are uncontrollable and the dashed lines are contingent links. The duration of how long it takes for the flood to block G is decided by the environment, which is not controllable by the scheduler, but it will be seen when the flood blocks G. The STPU uses an estimated lower and upper bounds to represent the contingent links such that the uncertain durations will be an arbitrary value within its bounds.

A *projection* of an STPU replaces each contingent link $e_{ij} \in [L_{ij}, U_{ij}]$ with a requirement link $e_{ij} \in [d, d]$ for some $L_{ij} \leq d \leq U_{ij}$. The resulting network represents a possible outcome of the uncontrollable choices and has no remaining uncertainty. A *schedule* of an STPU is an assignment of values to all time points. The schedule is consistent iff this assignment satisfies the bounds of all requirement links.

The solution to an STPU is an execution strategy. An *execution strategy*, S, maps projections of the STPU to schedules. S is *valid* iff $S(\pi)$ is consistent for every projection π. Because the durations of contingent links are unknown before execution, an execution strategy may be unsuccessful although it is valid. The definitions of "successful" execution strategies, called controllability, will be discussed in section 2.2.

In the real situation of evacuation planning, when the flood will block each road and how long each traffic flow drives through its routes are not controllable by the

decision maker. These uncertainties are ignored by the solution to the STN in Figure 2.1, but can be represented by the STPU example shown in Figure 2.3. The red nodes represent uncertain timepoints and the red and dashed links represent contingent links. We give a solution to the problem, which is a strategy, without providing how to generate such a solution because the approaches to solve STPU will be discussed in Section 2.2; The solution is: Region A will evacuate at $t = 0$ and Region B will evacuate 5 time units after the observation of Region A arriving at G (which is at $T_o \in [50, 70]$), so that the temporal difference from Region A passing G ($T_o + [30, 35]$) to Region B arriving at G ($T_o + 5 + [25, 30]$) is within $[-5, 5]$; The latest time of when Region B passing G is $T_o + [60, 70] \leq 140$.

2.1.1.1 Probabilistic STN (pSTN)

It is a natural extension to the STPU model to associate probabilities with the outcome (timing) of uncontrollable events [Tsamardinos, 2002; Fang et al., 2014]. The pSTN redefines the representations of the contingent links. Instead of the lower and upper bounds of the links, pSTN uses probability distributions to represent the temporal uncertainty. The **uncertain duration** $D_{ij} = t_j - t_i : \Omega \rightarrow R$ is a random variable describing the duration of the contingent link. For instance, the contingent link representing the traffic flow of region A drives from A to G in Figure 2.3 can be replaced by a normal distribution $\mathcal{N}(\mu, \sigma^2)$, where $\mu = 60$ and $\sigma = 10$.

From the probabilistic STN, we can measure the robustness of schedules and temporal plans by calculating the probability of success. Furthermore, the probability of failure can be treated as a constraint, and other more important preferences can be the objective [Fang et al., 2014; Wang and Williams, 2015].

2.1.2 Conditional Temporal Problem (CTP)

In temporal problems, not all timepoints and temporal constraints are always required, some of them are invoked under certain circumstances. Thus, temporal reasoning models describing conditions that activate or deactivate those objects are introduced. The Conditional Temporal Problem is one of those models that contain uncontrollable conditions.

Definition 2.3. A **Conditional Temporal Problem (CTP)** [Tsamardinos et al., 2003] is a tuple $< V, E, P, L, OV, O >$, where:

- V is a set of nodes,
- E is a set of constraints between nodes in V,
- P is a finite set of propositions,
- L is a function attaching a label which is a subset of P, to each node,
- $OV \subseteq V$ is the set of observation nodes,

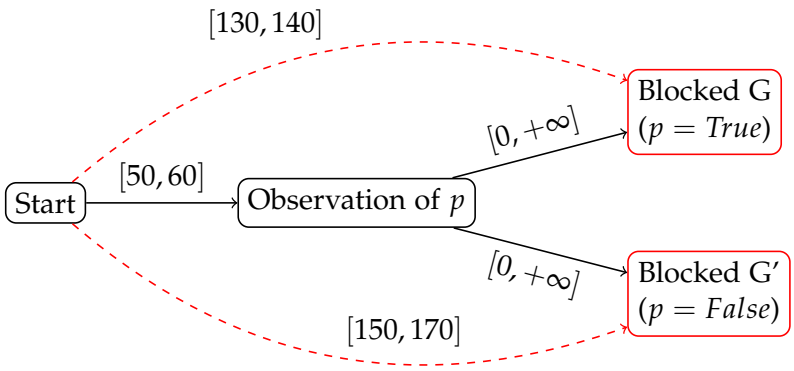


Figure 2.4: A CTP example

- $O : P \rightarrow OV$ is a projection associating a proposition with an observation node.

In a CTP, the values of propositions are not controllable but observable. Their truth value can be observed at their observation nodes. After the observations, nodes and links connected to the nodes will be activated or deactivated according to their labels. All constraints of a CTP can be disjunctive as the constraints in the TCSP. The authors also define the Conditional Simple Temporal Problem (CSTP), in which the constraints are binary ($l \leq y - x \leq u$) without disjunctions.

A CTP example of the evacuation planning is shown in Figure 2.4, the flood will either block G or G', which is an unknown condition depends on the environment. If the observation node of proposition p shows *True*, the node representing that G will be blocked and related links are activated, otherwise the node representing that G' will be blocked and related links are activated.

To discuss the solution to a CTP, the following definitions are given. Because the focus of this thesis is on the STPU and CCTPU, which has controllable conditions, we do not give formal definitions of the solutions to the CTP.

An *execution scenario* sc of CTP is a set of observed results of P. A *scenario projection* of the CTP in execution scenario sc, denoted as $Pr(sc)$, is a temporal problem $< V_1, E_1 >$, where V_1 and E_1 are the activated nodes and links according to sc. A *schedule* S of a CTP is a mapping $V \rightarrow \Re$, i.e. a time assignment to the nodes in V, denoted with $T(v)$. An *execution strategy* ES for a CTP is a function from the set of scenarios for a CTP to a schedule ES: $SC \rightarrow S$. A viable execution strategy is one such that $ES(sc)$ is a solution to the projection $Pr(sc)$ for each scenario $sc \in SC$. The execution strategy ES for a CTP is the solution to the problem.

We discuss the CTP to illustrate the intention that conditions represented by propositions can activate or deactivate subnetworks of the problem. In the technical parts of the thesis, we only consider controllable conditions, related works about uncontrollable conditions can be found in Section 6.3.

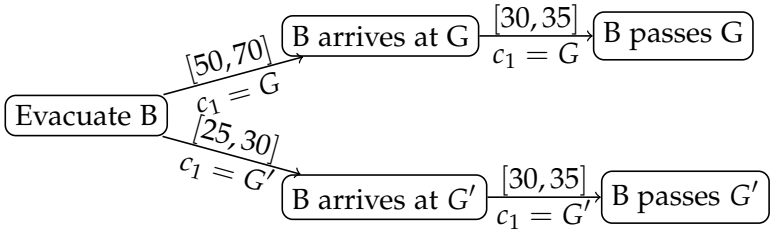


Figure 2.5: A CCTP example

2.1.3 Controllable Conditional Temporal Problem

The Controllable Conditional Temporal Problem (CCTP) was introduced to model temporal problems with controllable options [Yu and Williams, 2013]. The CCTP also has conditions that can activate or deactivate subnetworks, however, the conditions are decisions among options that can be decided by the scheduler. Furthermore, the conditions are modelled by assignments to discrete variables.

The CCTP model without objective functions (such as minimising the relaxation cost) is shown in the following definition.

Definition 2.4. A **CCTP** is a tuple $< V, E, C, D, \ell >$ where:

- $< V, E >$ is a temporal problem including time points and constraints,
- C is a set of discrete variables with finite domains,
- D is the collection of domains of C,
- ℓ is a mapping that attaches to each link a label including assignments to variables in C.

We also give a CCTP example of the evacuation planning problem which is shown in Figure 2.5. The example describes that the region B can evacuate through G or G', which is a controllable option represented by the assignment of discrete variable c_1. The domain of c_1 is $\{G, G'\}$. The links representing constraints in each option are attached with labels of the assignments $c_1 = G$ or $c_1 = G'$.

The application in [Yu and Williams, 2013] is to resolve over-constrained traffic plans since the requirements from the users are usually too tight to be satisfied. Relaxing the over-constrained CCTP problem, also adds the following terms to the CCTP: $RE \in E$ is a set of relaxable temporal constraints, $f_p : D(c) \rightarrow \Re^+$ is a function calculating rewards of each assignment and $f_e : (e, e_r) \rightarrow \Re^+$ is another function calculating relaxation costs for each relaxable links, where e_r is the relaxed bound of e. Therefore, the solutions for over-constrained CCTP problems can be represented by a pair $< A, R >$, where A is a set of complete assignments to C and R is a set of relaxations to RE that make the CCTP consistent.

The authors presented a Best-first Conflict-Directed Relaxation algorithm that enumerates the relaxations to an over-constraint CCTP in best-first order. By learning

conflicts through negative cycles and relaxing conflicts through extending temporal constraints in RE, the BCDR can find consistent candidates. And the core process is the Conflict-directed A^* algorithm.

Moreover, a general way to represent the solution to a CCTP is a pair $< A, S >$, where A is a complete assignment to C and S is a consistent schedule of the STN from the CCTP by activating or deactivating links by A.

2.1.4 Controllable Conditional Temporal Problem with Uncertainty

In order to provide solutions that can be executed more robustly, CCTP with uncertainty was introduced to solve over-constrained temporal problems with uncertainty and choices [Yu et al., 2014].

Definition 2.5. A **Controllable Conditional Temporal Problem with Uncertainty (CCTPU)** is a 5-tuple $< V, E, C, D, \ell_E >$, where

- $< V, E >$ is an STPU,
- C is a set of controllable discrete variables,
- $D(c)$ is the domain of variable $c \in C$,
- ℓ_E is a mapping that attaches to each link in E a (possibly empty) conjunction of assignments to variables in C.

The CCTP is a special case of CCTPU where $V_U = \emptyset$. Note that when $C = \emptyset$ the CCTPU reduces to an STPU.

For the over-constrained CCTPU, Yu et al. [2014] defined a relaxation solution as a pair $< A, R >$, where A is a complete assignment of C and R is the set of relaxations that can make the problem satisfy a certain set of constraints, for instance, dynamic controllability which we will discuss in the next section.

The CCTPU is the problem we will solve in Chapter 4, an illustrative example and detailed definitions will be addressed in that chapter.

2.2 Dynamic Controllability of The STPU

Because of the uncertainty, a consistent STPU may fail during execution. Vidal and Fargier [1999] introduced the weak, dynamic and strong controllability to describe different levels of successful abilities when executing an STPU. Among the three levels of controllability, the dynamic controllability is the most useful, flexible and challenging one. In this section, we present the concepts of the different levels of controllability and the dynamic controllability checking algorithms that help to understand the following chapters.

2.2.1 The Three Levels of Controllability of the STPU

Based on the definition of execution strategy we mentioned in section 2.1.1, which is a projection from uncertain situations to schedules, we review the following definitions.

Definition 2.6. An STPU is **weakly controllable**, iff for all uncertain situations π of the STPU there exists a schedule S, such that $S(\pi)$ is consistent.

In other words, different schedules can be found if we know which uncertain situation the contingent links may be. However, in most real applications, the uncertain situations cannot be predicted precisely before executions, which makes the weak controllability not very useful.

Definition 2.7. An STPU is **strongly controllable**, iff there exists a schedule S such that for all uncertain situations π of the STPU $S(\pi)$ is consistent.

A strongly controllable STPU has a universal schedule that can deal with any uncertain situations represented by the problem. The strong controllability is useful for many problems. For example, given a fixed schedule or plan generated from a solver that does not consider the uncertainty, the fixed solution can be treated as a strongly controllable solution with zero uncertainty. Furthermore, the question how much uncertainty the solution can absorb may be answered by adding uncertainty incrementally until the solution cannot satisfy the strong controllability condition. Although the strong controllability is useful, sometimes it is too strict to be satisfied. Therefore, the dynamic controllability was introduced.

Definition 2.8. Given a schedule, T, and a specific time t, $T_{<t}$ is the restriction of T to all time points scheduled before t. An STPU is **dynamically controllable** iff there exists a valid execution strategy S such that $S(\pi_1)_{<t} = S(\pi_2)_{<t}$ implies $S(\pi_1)(x) = S(\pi_2)(x)$, where $t = S(\pi_1)(x)$ Hunsberger [2009], for all projections π_1, π_2 and executable time point x.

This means that the time the strategy assigns to the executable point x can only depend on uncontrollable durations observed earlier. An execution strategy like that in definition 2.8 is called a "dynamic execution strategy".

The dynamic controllability is applicable as long as the uncertain situations can be observed when finished. It is also more flexible so that the problems that have no universal solutions may have dynamically controllable strategies to guide a successful execution under uncertain situations.

2.2.2 Checking Dynamic Controllability for the STPU

Different algorithms that can verify if a given STPU is dynamically controllable have been well studied. The approaches in this thesis are mainly based on Morris's algorithms which will be discussed in this subsection. Besides Morris's cubic algorithm,

Hunsberger [2013] introduced another $O(n^3)$ method, and Nilsson et al. [2014] introduced an $O(n^3)$ incremental algorithm based on Shah et al. [2007]'s incremental algorithm, which are the most efficient approaches.

In this section, we briefly review the developments of dynamic controllability checking algorithms and discuss the first polynomial algorithm, which we call the classic algorithm, in section 2.2.2.1, and Morris's cubic algorithm in section 2.2.2.2 in detail. We select these two because the constraint model in Chapter 3 is built based on the classic algorithm and Chapter 4 is based on the cubic algorithm.

The first dynamic controllability checking algorithm was introduced by Vidal and Fargier [1999] with the original definition of dynamic controllability of STPU. The approach is a two-player game – one decision agent plays against the other nature agent. An STPU is dynamically controllable if and only if the decision agent can always make successive decisions based the past no matter what the nature agent has done.

2.2.2.1 Classic Algorithm

The first polynomial algorithm for checking dynamic controllability of the STPU was introduced by Morris et al. [2001], based on *boundary projection* and *safe networks* [Morris and Muscettola, 2000]. This algorithm repeatedly tightens requirement links and introduces wait constraints according to the triangular reduction and wait reduction (which are discussed later) and checks if the tightened network is pseudo-controllable until no link can be tightened or the network is not pseudo-controllable. If the tightened requirements on a pair of variables connected by a contingent link are tighter than the contingent links' bounds, the contingent link is said to be "squeezed". In this case, the network is not pseudo-controllable.

The triangular reduction repeatedly examines each triangle of time points in the STPU, considering at most one contingent link each time. Figure 2.6 shows a triangle, with time points A, B and C. The link between A and C is contingent, the other two are requirement links. (If e_{AB} is also contingent, it is considered in a separate triangle.) First, the algorithm applies the implied (shortest path) bounds (e.g., $L_{BC} \leftarrow \max(L_{BC}, L_{AC} - U_{AB})$ and $U_{BC} \leftarrow \min(U_{BC}, U_{AC} - L_{AB})$). The next step depends on the relation between B and C:

- If $U_{BC} < 0$, B must be scheduled after C (hence, after C has been observed), so no further adjustments are needed. This is called the "follow" case.

- If $L_{BC} \geq 0$, B must be scheduled before or simultaneously with C (i.e., before C has been observed). This is called the "precede" case, and the bounds on the e_{AB} link are updated to $L_{AB} \leftarrow \max(L_{AB}, U_{AC} - U_{BC})$ and $U_{AB} \leftarrow \min(U_{AB}, L_{AC} - L_{BC})$.

- If $L_{BC} < 0$ and $U_{BC} \geq 0$, B may be scheduled before or after C. This case, called the "unordered" case, introduces a conditional bound, called a "wait", $\langle C, U_{AC} - U_{BC} \rangle$ on e_{AB}, with the meaning that execution of B must wait for

either C to occur or at least $U_{AC} - U_{BC}$ after A. If $U_{AC} - U_{BC} \leq L_{AC}$, C cannot occur before the wait expires, so the wait is replaced by the unconditional bound $L_{AB} \leftarrow \max(L_{AB}, U_{AC} - U_{BC})$. Otherwise, $L_{AB} \leftarrow \max(L_{AB}, L_{AC})$, since the wait for C will delay B to at least L_{AC} after A.

Tighter bounds on a requirement link propagate to any other triangle that the link is part of.

In addition, the algorithm performs "wait regression", in which conditional bounds are propagated to other links (Figure 2.7). If $\langle C, w \rangle$ is a wait on e_{AB}, where $w \leq U_{AC}$, then (i) if there is any link e_{DB} with upper bound U_{DB}, a wait $\langle C, w - U_{DB} \rangle$ is added to e_{AD}; and (ii) if $w \geq 0$ and there is a contingent link e_{DB}, where $B \neq C$, with lower bound L_{DB}, a wait $\langle C, w - L_{DB} \rangle$ is added to e_{AD}.

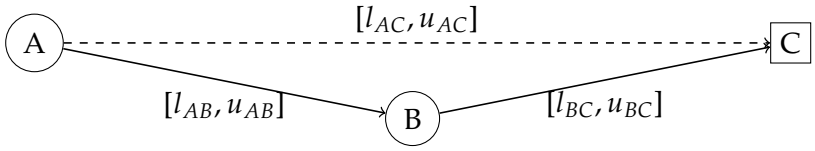


Figure 2.6: Triangular reduction: (1) precede case ($l_{BC} \geq 0$): $l_{AB} \leftarrow u_{AC} - u_{BC}$, $u_{AB} \leftarrow l_{AC} - l_{BC}$; (2) unorder case ($l_{BC} < 0$ and $u_{BC} \geq 0$): wait constraint $w_{AB} \leftarrow < C, u_{AC} - u_{BC} >$; (3) follow case ($u_{BC} < 0$).

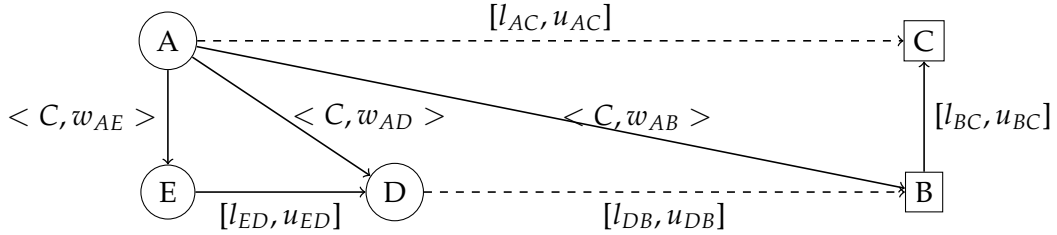


Figure 2.7: Wait Reduction: (1) Reduction through the contingent link: $w_{AD} \geq w_{AB} - l_{DB}$; (2) Reduction through the requirement link: $w_{AE} \geq w_{AD} - u_{ED}$; (3) Wait Bounds: $l_{AX} \geq min(l_{AC}, w_{AX})$.

2.2.2.2 Advanced Verification Algorithms

Morris and Muscettola [2005] provided an $O(n^5)$ algorithm by representing the STPU by a *labelled distance graph* and obtaining cutoff bounds analogous to the Bellman-Ford structure instead of exhausting the tightening. In the *labelled distance graph*, the requirement links are formulated in the same way as in the distance graph introduced by Dechter et al. [1991]. Each link $A \xrightarrow{[l,u]} B$ is represented by two directed links $A \xrightarrow{u} B$ and $B \xrightarrow{-l} A$. The contingent links are represented with labelled edges.

Each link $A \xrightarrow{[l,u]} B$ is presented by $A \xrightarrow{b:l} B$ and $B \xrightarrow{B:-u} A$, which are called the lower-case and upper-case edges respectively. The lower-case edges represent the case that the environment decides the constraint should take on the minimum length and the upper-case edges represent the case where it takes the maximum length. Thus, the reduction rules of dynamic controllability can be more uniform and the reduction rules in Morris and Muscettola [2005] are still used in the following and more efficient algorithms.

The list of advanced reduction rules and a table showing what the precede/follow/unorder case reductions in section 2.2.2.1 look like in the advanced reduction rules will be illustrated after introducing the next improvement of the reductions.

In 2006, Morris introduced an $O(n^4)$ algorithm. The author first rephrased the reduction rules by transforming the labelled edges for contingent link $A \xrightarrow{[l,u]} B$ to $A \underset{l}{\overset{-l}{\rightleftharpoons}} A' \underset{b:0}{\overset{B:l-u}{\rightleftharpoons}} B$. Thus, the label removal condition $x \geq l^c$ can be changed to $x \geq 0$. The reduction rules are shown in Equation 2.1.

$$
\begin{aligned}
&\text{(Upper-case Reduction)}\\
&A \xleftarrow{B:x} C \xleftarrow{y} D \text{ adds } A \xleftarrow{B:(x+y)} D.\\
&\\
&\text{(Lower-case Reduction) If } x < 0,\\
&A \xleftarrow{x} C \xleftarrow{c:y} D \text{ adds } A \xleftarrow{x+y} D.\\
&\\
&\text{(Cross-case Reduction) If } x < 0, B \neq C,\\
&A \xleftarrow{B:x} C \xleftarrow{c:y} D \text{ adds } A \xleftarrow{B:(x+y)} D.\\
&\text{(No-case Reduction)}\\
&A \xleftarrow{x} C \xleftarrow{y} D \text{ adds } A \xleftarrow{x+y} D.\\
&\\
&\text{(Label Removal) If } x \geq 0\\
&A \xleftarrow{B:x} C \text{ adds } A \xleftarrow{x} C.
\end{aligned}
\tag{2.1}
$$

A mapping from the reduction rules of the classic algorithm to the reduction rules of the advanced algorithm is given in Table 2.1.

	Classic Algorithm	Advanced Algorithms
Precede Case	$u_{AB} \leftarrow l_{AC} - l_{BC}$	Lower-Case Reduction
	$l_{AB} \leftarrow u_{AC} - u_{BC}$	Upper-case Reduction, Label Removal
Unorder Case	$w_{AB} \leftarrow < C, u_{AC} - u_{BC} >$	Upper-case Reduction
	$l_{AX} \geq min(l_{AC}, w_{AX})$	Label Removal
	$w_{AD} \geq w_{AB} - l_{DB}$	Cross-Case Reduction
	$w_{AE} \geq w_{AD} - u_{ED}$	Upper-case Reduction
Follow Case	Shortest Path Reductions	No-case Reduction

Table 2.1: The mapping of reduction rules

The conditions in the reduction rules (Equation 2.1) are about checking if the weights are negative or not, which is the only difference from the rules in Morris and Muscettola [2005] whose condition in the Label Removal (see Equation 2.2) depends on two numbers. This little difference helps to introduce a faster propagation algorithm.

$$\begin{aligned}&\text{(Label Removal [Morris and Muscettola, 2005]) If } x \geq -z \\ &B \xleftarrow{b:z} A \xleftarrow{B:x} C \text{ adds } A \xleftarrow{x} C.\end{aligned} \tag{2.2}$$

Using these reduction rules, a sequence of labelled edges can be transformed into a new edge. Furthermore, if a reduced path does not have lower-case edges by a sequence of reductions, it is *semi-reducible*.

Theorem 2.1. *[Morris, 2006] An STPU is Dynamically Controllable if and only if it does not have a semi-reducible negative cycle.*

In this thesis, we call these semi-reducible negative cycles dynamic controllability conflicts or conflicts. With Theorem 2.1, a faster dynamic controllability checking algorithm was introduced, which propagates lower-case edges with Dijkstra's algorithm and checks for semi-reducible negative cycles with the Bellman-Ford algorithm iteratively. The number of iterations is no more than the number of contingent links, so the algorithm is $O(n^4)$.

The current fastest algorithms are $O(n^3)$ [Morris, 2014; Hunsberger, 2013; Nilsson et al., 2014]. Morris's new method propagates negative links backwards in order to find potential *moat edge*, which is the first edge e' following a lower-case edge e in path P and its reduced distance $D_P(end(e), end(e'))$ is negative. The key process in this algorithm is called *DCbackprop* that propagates the end of a negative link backwards in Dijkstra's algorithm while applying the reduction rules of dynamic controllability. When meeting the end of another negative link, it calls *DCbackprop* again. Otherwise, it sums up the weights of paths until they are not negative. The non-negative paths are reduced as new edges without a lower-case label.

Theorem 2.2. *[Morris, 2014] The DCbackprop procedure encounters a recursive repetition if and only if the STPU is not Dynamically Controllable.*

Theorem 2.2 is proved by Morris [2014]. The proof proves that the DCbackprop procedure encounters a recursive repetition if and only if the STPU has a semi-reducible negative cycle. Therefore, a dynamically controllable STPU can pass the checking algorithm without early terminations caused by a conflict.

2.2.2.3 The Strong Controllability Reduction Rules

The strong controllability checking algorithm [Vidal and Ghallab, 1996] is polynomial because the verification of strong controllability is a local process. In other words, verifying strong controllability do not have the regression process. To compare the reduction rules of strong controllability and dynamic controllability, the following equations illustrate in the form of labelled distance graph.

(SC-Upper-case Reduction)
$A \xleftarrow{B:x} C \xleftarrow{y} D$ adds $A \xleftarrow{x+y} D$.

(SC-Lower-case Reduction)
$A \xleftarrow{x} C \xleftarrow{c:y} D$ adds $A \xleftarrow{x+y} D$.

(No-case Reduction)
$A \xleftarrow{x} C \xleftarrow{y} D$ adds $A \xleftarrow{x+y} D$.

2.3 Partial Order Schedules and Robustness Measures

In this section, we review four existing robustness measures of temporal problems with which we are going to compare in the thesis. These robustness measures can be used to evaluate Partial Order Schedules.

2.3.1 Partial Order Schedules

The Partial Order Schedules (POS) are partial solutions to scheduling problems. A scheduling problem consists of a set of activities that will be scheduled, a set of temporal constraints among those activities that have to be satisfied, a set of different resources with limited capacities and a set of resource constraints that describe how much each kind of resource each activity occupies during execution. The solution to the scheduling problem is to find a feasible schedule that can assign a proper start time for each activity such that all constraints are satisfied.

Instead of giving start times for activities, a POS partially solve scheduling problems by adding temporal constraints to resolve resource constraints.

Definition 2.9. [Policella et al., 2004] A **Partial Order Schedule** for a problem is a graph, where nodes are the activities of the problem and the edges represent temporal constraints between pairs of activities, such that any possible temporal solution is also a consistent assignment.

For instance, an example of the Resource-Constrained Project Scheduling Problems with minimum and maximum time lags (RCPSP/max) [Bartusch et al., 1988], is shown in Figure 2.8. The RCPSP/max example has two activities to be scheduled and each of them will occupy 3 units of resource I, which will make a conflict to schedule them overlapped with each other, since the total capacity of the resource I is 5. The durations of the two activities are 5 minutes.

The solid line is a temporal constraint of the problem, which means the difference between the end times of the activities is no larger than 10 minutes. The dashed line is the added constraint that resolves the resource conflict, which means if activity II is scheduled after the completion of activity I, the resource constraint is always satisfied.

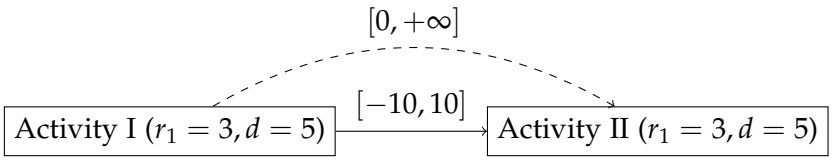


Figure 2.8: An example of the resource-constrained project scheduling problem has 5 units of resource I.

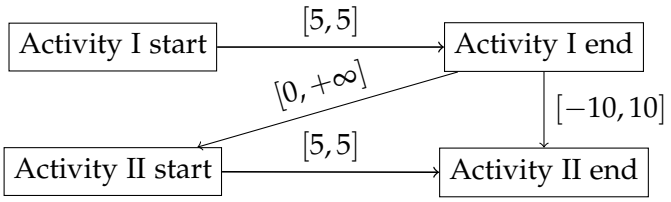


Figure 2.9: The STN of the POS in Figure 2.8.

The POS can be represented by an STN by using two nodes and one link to represent an activity. Figure 2.9 shows the STN of the example.

By adding additional temporal constraints, the POS resolves all the resource constraints and maintains the original temporal constraints from the problem. A POS represents a set of consistent schedules of the scheduling problem. The decisions among these consistent schedules can be made during execution, which enables the system to be flexible to deal with unexpected disruptions. Therefore, measuring and optimising the flexibility of POS are interesting research topics. In the rest of this section, we will list several robustness measures.

2.3.2 Flexibility

The metric flexibility was introduced by Aloulou and Portmann [2003]. This measure counts the number of pairs that do not have any explicit or implicit precedence relations so that the decisions of the orders of such pairs are flexible. Policella et al. [2004] used this metric to measure the flexibility of the POS. The definition of $flex$ is

$$flex = \frac{|\{(a_i, a_j) | a_i \not\prec a_j \wedge a_j \not\prec a_i\}|}{n(n-1)}, \tag{2.3}$$

where the precedence constraints between activities include both explicit and implicit relations. Thus, the higher $flex$ the POS gets, the lower degree of interaction among activities it achieves.

The measure flexibility tries to calculate how many real schedules may be contained in a POS. However, in the examples shown in Figure 2.10, the graph with higher $flex$ has fewer solutions. Both graphs in the figure have 6 nodes representing 6 activities, graph (a) has 6 pairs of activities that do not have any precedence and

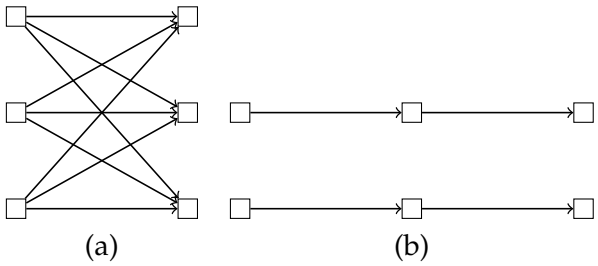


Figure 2.10: Examples showing flexibility

graph (b) has 18 pairs. Thus, according to the measure of flexibility, graph (b) is more flexible than graph (a). However, the nodes in graph (a) can be ordered in 36 different ways that do not violate any precedences, but the nodes in the graph (b) have only 20 ways.

Therefore, the $flex$ implies robustness-related features, such as the independence, however, it does not consider different kinds of flexibility in a POS, it even cannot present how many different solutions contained in the POS. Additionally, if a POS has low $flex$ but every time constraint is loose enough to allow a certain range of deviation, the $flex$ is not able to show its advantages.

2.3.3 Fluidity

Some metrics take temporal slacks into account when measuring robustness [Cesta et al., 1998; Hunsberger, 2002; Boerkoel and Durfee, 2013]. To show the difference from flexibility $flex$, we use the name fluidity $fldt$ which was introduced in [Cesta et al., 1998]. It represents the ability to absorb temporal deviations. It is defined as

$$fldt = \sum_{i=1}^{n} \sum_{j=1 \wedge j \neq i}^{n} \frac{slack(a_i, a_j)}{H \times n \times (n-1)} \times 100 \tag{2.4}$$

where H is a fair bound which is large enough to allow all activities to be executed, and $slack(a_i, a_j)$ is the width of the allowed distance interval between two activities a_i and a_j. The higher the $fldt$ is, the lower the risk of cascading change, the higher tolerance to temporal deviations, and the higher the probability of changes in response to disruption remaining local.

However, the metric fluidity does not always illustrate the ability to absorb temporal deviations correctly. As the examples are shown in Figure 2.11, both examples have three activities and two precedences among them. The temporal slacks of the precedence constraints are the same, which means that they have the same fluidity. If the duration of activity A is longer than expected, i.e. if there is a delay, it requires the same amount of flexibility from link AB as the delay in the example (a) but the same amount of flexibility from link AB and AC as the delay in the example (b). In other words, the delay of activity A will cost twice the temporal slacks in example (b) than in (a). Therefore, the two examples have different abilities to absorb temporal

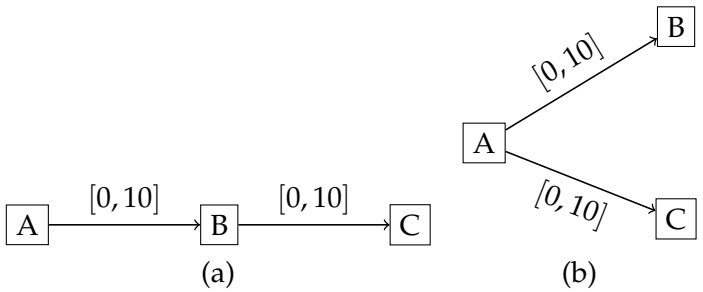


Figure 2.11: Examples showing fluidity

deviations, but the $fldt$ is not able to tell the difference.

2.3.4 Disruptability

This measure called disruptability was introduced by Policella et al. [2004]. It considers stability against changes. The definition of $dsrp$ is

$$dsrp = \frac{1}{n}\sum_{i=1}^{n}\frac{slack(a_i)}{num_{changes}(a_i, \Delta a_i)} \tag{2.5}$$

The $slack$ here is different from the one used in $fldt$. It represents the temporal variability of a single activity and equals the difference between the upper bound and the lower bound of the end time of activity a_i. The function $num_{changes}(a_i, \Delta a_i)$ counts the number of activities that are changed in the process of right-shifting activity a_i by an amount of time Δa_i which is a constant number no larger than $slack(a_i))$. This measure takes disruption into account and calculates the influence of time delay of activities. The essence of this measure estimates the trade-off between flexibility and the implied changes.

2.3.5 Improved Fluidity

Wilson et al. [2014] proposed an improved measure of the total temporal tolerance of an STN. The authors thought the previous $fldt$ overestimated the robustness by calculating dependent slacks repeatedly. For instance, suppose that a sequence of activities need to be done one by one and the total time should be within an absolute upper bound. Then the temporal slack of the whole sequence will be calculated as many times as the number of activities using the original fluidity measure.

The improved fluidity measure treats every activity as two separated nodes which are the start and the end nodes, and every initial constraint as an end-to-start constraint. The fluidity is optimising the sum of the slack between the start and end nodes of each activity, which satisfies all the temporal constraints.

By solving the following LP model, we can get the improved fluidity.

$$\begin{aligned} max\ & \sum_{t \in T} (end(t) - start(t)) \\ s.t.\ & start(t) \leq end(t) && t \in T \\ & start(t_i) - end(t_j) \leq c && \forall (t_i - t_j \leq c) \in C \end{aligned} \tag{2.6}$$

In the formula, T and C are the sets of activities and constraints in the original STN separately and $end(t) - start(t)$ is the slack of the time for scheduling activity t.

This measure improves on the original definition of the fluidity by considering dependency between the activities. And it can be extended to the STPU as well. Wilson et al. [2014] introduced a model to calculate the fluidity of an STPU subject to strong controllability constraints. The difference in calculating this fluidity measure between the STN and STPU is that only the slacks of controllable nodes are considered in the objective function.

The improved fluidity shows a way to broaden the flexibility rather than sum up the real factors of the temporal network. It calculates the implicit temporal tolerance by solving an LP model. Mountakis et al. [2015] showed that the improved fluidity could be computed in $O(n^3)$ instead of solving the LP model.

2.3.6 Summary

Different robustness measures evaluate the quality of a schedule from different views. However, whether the "good" solution appraised by each measure is perfect and whether a better solution according to one measure will still be the better one according to other measures, are not guaranteed. For instance, a POS with too much flexibility will not achieve a high value of disruptability and a POS with high fluidity may get low flexibility as well.

Furthermore, regardless of what specific features those measures exactly take into account, they calculate the average level across the schedule, which means they ignore the deviation in detail. Specifically, if there is a *weak point* in the POS such as a link with a tight time constraint or a small group of activities with strong precedence constraints, then it is still possible to achieve an excellent average value when the rest of the schedule is strong enough. Therefore, we need an in-depth view of the relationships and differences among the measures. And a better model to calculate robustness of schedules and temporal plans is worth exploring.

Chapter 3

Optimising STPU

The Simple Temporal Problem with Uncertainty (STPU) [Vidal and Fargier, 1999], is a widely used model for representing schedules or temporal plans that have both uncertainty about the timing of some events (for example, the time needed to complete an activity) and flexibility for the executing agent to choose the timing of other events (for example, the time to start an activity).

An STPU, requires *controllability*, meaning, informally, that the executing agent has a valid (constraint-satisfying) response to any choice by the environment. In this chapter, we will formulate dynamic and strong controllability (Section 2.2.2) constraint models. If the agent can make the choices before having any observation, the network is strongly controllable. The more practically useful, but also more complex, the property of dynamic controllability means the agent can choose a value for each controllable time point variable using only observations of uncontrollable events that have taken place earlier, such that all constraints will be respected under any outcome of future uncertainties.

It is known that deciding if a *given* STPU is dynamically controllable [Morris et al., 2001] or strongly controllable [Vidal and Ghallab, 1996] can be done in polynomial time. The problem that we consider in this chapter is optimising an objective function over the bounds on time point differences, subject to the constraint that the network is dynamically controllable or strongly controllable. This has a broad range of applications. As illustrative examples, we consider minimally relaxing an over-constrained (non-controllable) STPU to make it dynamically controllable [Yu et al., 2014]; finding the minimum schedule flexibility needed to maintain dynamic controllability [Wah and Xin, 2004]; maximising different measures of schedule robustness; and optimising a preference function over a probabilistic STN with chance constraints [Fang et al., 2014].

The problem of optimising time bounds under dynamic controllability was previously considered by [Wah and Xin, 2004], who formulated a non-linear constraint optimisation model. In fact, dynamic controllability is a disjunctive linear constraint, and using this insight we consider several alternative ways of dealing with it, including a conflict-driven search [Yu et al., 2014], a formulation as a mixed-integer linear program with 0/1 variables, and the non-linear encoding proposed by Wah and Xin.

Besides the constraint model of dynamic controllability, this chapter also intro-

duces a constraint model of strong controllability of STPU. The optimisation problem of STPU under strong controllable constraints can measure the quality of fixed temporal plans and schedules because fixed solutions are strongly controllable. Thus, the robustness metrics can be applied to current planning and scheduling systems that generate fixed solutions. Furthermore, checking strong controllability for STPU is polynomial [Vidal and Fargier, 1999; Vidal and Ghallab, 1996]. Strong controllability requires a solution that can deal with all uncertain situations, so worst cases among those uncertain situations are the only necessities to be tested. In STPU, all worst cases can be enumerated by a combination of lower or upper bounds of contingent links. The constraint model of strong controllability can also be generated by its reduction rules, which is a linear programming model. Therefore, the constraint model of strong controllability can (1) be used to formulate robustness measures of temporal plans and schedules based on optimisation problems and (2) be compared with the constraint model of dynamic controllability.

In this chapter, we first introduce the optimisation model of STPU under controllability constraints. After the general formulation, the constraint model of dynamic controllability is introduced, which is a disjunctive linear constraint model that can be encoded by a Mixed Integer Programming (MIP) model or a Non-linear Programming model. This model can also be solved by a Conflict-directed Relaxation with Uncertainty (CDRU). The constraint model of strong controllability follows the constraint model of dynamic controllability, which is used in the experiments to compare the improvement from strong controllability to dynamic controllability. Then, we provide five application problems to illustrate the differences among those approaches and evaluate the improvements from strong controllability to dynamic controllability. The main work in this chapter has been published in a conference paper [Cui et al., 2015].

3.1 Problem Formulation

The general form of the optimisation problem can be stated as follows: We are given the structure of an STPU, that is, the set of time points $V = V_E \cup V_U$ and links $E = EC \cup EU$, but not the upper and lower bounds on (all) links, and an objective function. The problem is then to set those bounds so as to optimise the objective function value:

$$
\begin{array}{ll}
\text{opt} & f_{obj}(l_{ij}, u_{ij} \mid e_{ij} \in E) \\
\text{s.t.} & L_{ij} \leq l_{ij} \leq u_{ij} \leq U_{ij} \\
 & N(l_{ij}, u_{ij} \mid e_{ij} \in E) \text{ is dynamically controllable or strongly controllable} \\
 & \text{application-specific side constraints}
\end{array}
$$

The decision variables, l_{ij} and u_{ij}, represent the lower and upper bounds on link e_{ij}. Thus, a satisfying assignment defines an STPU, $N(l_{ij}, u_{ij} \mid e_{ij} \in E)$, and this STPU must be dynamically controllable or strongly controllable. The main contribution of this chapter is the formulation of dynamic controllability as a set of disjunctive linear

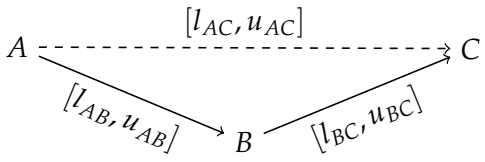


Figure 3.1: An STPU triangle. The A–C link is contingent.

constraints $DC*$. We then show how these constraints can be reformulated into a mixed-integer linear program (MIP) or a non-linear program (NLP), which can be given to different optimisation solvers.

L_{ij} and U_{ij} are constants, which constrain the range of the bounds variables. In principle, these constants are not needed. They can be set to $-\infty$ and ∞, respectively, leaving the bounds variables unrestricted. Many of the application problems we consider (cf. Section 3.4), however, specify narrower ranges, and we use this to simplify the constraint formulation.

This is a general formulation, which can allow for the bounds of any link to change. If in a particular application only some links are modifiable, this is modelled by fixing the bounds of the non-modifiable links with equality constraints.

3.2 Constraint Model of Dynamic Controllability

Checking dynamic controllability of an STPU was first shown to be tractable by Morris et al. [2001]. Their algorithm repeatedly applies a set of reductions, tightening the bounds on requirement links, until no more reductions apply (in which case the network is controllable) or the network becomes inconsistent (implying it is not controllable). The reduction rules can be found in Section 2.2. Our constraint model uses mainly the same reduction rules but in the form of constraints between the decision variables that represent link bounds.

3.2.1 Disjunctive Linear Model

Constraints are formulated over each triangle of nodes in the STPU, considering at most one contingent link each time.

Shortest path constraints Every triangle in the STPU has to satisfy the shortest path constraints to maintain the consistency of the network. Regardless of the uncertainty of contingent links, every triangle can be transformed into the distance graph as shown in Figure 3.2. By using the consistency checking method of STN, in a triangle, an edge has to be no larger than the two-edge path with the same pair of start and

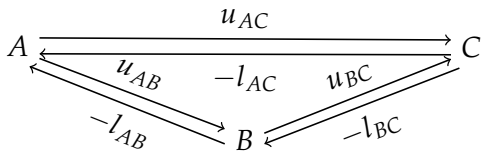


Figure 3.2: The distance graph of a triangle.

end nodes. Otherwise the edge must be shrunk to make the network consistent.

$$\begin{array}{rcl} -l_{AC} & \leq & -l_{AB} + -l_{BC} \\ -l_{AB} & \leq & -l_{AC} + u_{BC} \\ -l_{BC} & \leq & -l_{AC} + u_{AB} \\ u_{AC} & \leq & u_{AB} + u_{BC} \\ u_{AB} & \leq & u_{AC} - l_{BC} \\ u_{BC} & \leq & -l_{AB} + u_{AC} \end{array}$$

Therefore, we can rewrite these shrotest path constraints by removing the negative signs as follows:

$$\begin{array}{rcccl} l_{AC} & \leq & u_{AB} + l_{BC} & \leq & u_{AC} \\ l_{AC} & \leq & l_{AB} + u_{BC} & \leq & u_{AC} \\ u_{AC} & \leq & u_{AB} + u_{BC} & & \\ & & l_{AB} + l_{BC} & \leq & l_{AC} \end{array} \tag{3.1}$$

The shortest path constraints can propagate in any direction (i.e., from the contingent link e_{AC} to the requirement links and vice versa.) This may seem contradictory, since a contingent link may not be squeezed by requirements. However, l_{AC} and u_{AC} here are variables, whose values will be the bounds on the contingent link. In some applications (e.g., the problem of minimising flexibility which motivated Wah and Xin) these variables are fixed to given constant values but other applications (e.g., problem relaxation) allow the bounds of contingent links to vary.

If no link in the triangle is contingent, these are the only constraints. Assuming, w.l.o.g., that the e_{AC} link is contingent, what constraints are needed is determined by the outer bounds on the e_{BC} link, following the cases explained in Section 2.2.2.2. If $U_{BC} < 0$, then $u_{BC} < 0$ and the triangle must always be in the follow case. Thus, no additional constraints are needed.

Precede constraints If $L_{BC} \geq 0$, then $l_{BC} \geq 0$ and the triangle will be in the precede case. The following constraints must hold:

$$\begin{array}{rcl} u_{AB} & \leq & l_{AC} - l_{BC} \\ l_{AB} & \geq & u_{AC} - u_{BC} \end{array} \tag{3.2}$$

This together with (3.1) is equivalent to

$$\begin{array}{rcl} u_{AB} & = & l_{AC} - l_{BC} \\ l_{AB} & = & u_{AC} - u_{BC} \end{array} \tag{3.2'}$$

since $l_{AB} \leq u_{AB}$ is always required. If $L_{BC} < 0$ and $U_{BC} \geq 0$, the triangle can be in any case, depending on the values given to l_{BC} and u_{BC}. The precede constraint then becomes disjunctive:

$$(l_{BC} < 0) \quad \vee \quad \left(\begin{array}{rcl} u_{AB} & \leq & l_{AC} - l_{BC} \\ l_{AB} & \geq & u_{AC} - u_{BC} \end{array} \right) \tag{3.3}$$

Triangular wait constraints If it is possible that the triangle may be in the unordered case ($L_{BC} < 0$ and $U_{BC} \geq 0$), a variable representing the conditional wait bound is added:

$$w_{ABC} \quad \geq \quad u_{AC} - u_{BC} \tag{3.4}$$

Regression of waits (described below) may introduce wait variables w_{ABX}, where X is any uncontrollable time point (not necessarily in the same triangle as A and B). For each requirement link e_{AB} and wait variable w_{ABX}, the following disjunctive constraint must hold:

$$l_{AB} \quad \geq \quad \min(l_{AX}, w_{ABX}) \tag{3.5}$$

If $U_{AB} \leq L_{AX}$, this simplifies to $w_{ABX} = l_{AB}$. The constraint $w_{ABX} \leq u_{AB}$ must also hold.

Wait regression Each wait bound $\langle X, t \rangle$ on a link e_{AB} is represented by a variable w_{ABX}. If there is a wait w_{ABX} and a contingent link e_{DB}, then wait regression implies the constraint

$$(w_{ABX} < 0) \quad \vee \quad \left(\; w_{ADX} \quad \geq \quad w_{ABX} - l_{DB} \; \right) \tag{3.6}$$

It is disjunctive because the regression only applies when $w_{ABX} \geq 0$. The weaker constraint

$$w_{ADX} \quad \geq \quad w_{ABX} - u_{DB} \tag{3.7}$$

holds in all cases (i.e., also when e_{DB} is a requirement link, or the wait is negative).

The wait decision variable w_{ABX} in this chapter is different from the wait reduction $< B, x >$ in Section 2.2.2.1. In the solution of the constraint model, the value of w_{ABX} is the maximum wait among triangular and regression waits $< B, x >$ in triangle ABX.

3.2.1.1 Correctness

The correctness of our constraint formulation can be shown to follow from that of the DC checking algorithm in Morris et al. [2001]. Their algorithm applies a set of reduction rules, which tighten the bounds on links, until quiescence; if the network at that point is consistent, the original network is dynamically controllable.

Let $N(l_{ij}, u_{ij} \mid e_{ij} \in E)$ be an STPU that is defined by a solution to our model,

i.e., constraints (3.1)–(3.7). In the detailed proofs, we will show that applying all the reduction rules from the DC checking algorithm to this STPU will not result in a tightening of any bound. That is, the STPU is already at quiescence. Since it must satisfy the shortest path constraints (3.1), it is also consistent. Thus, the DC checking algorithm applied to N will report that it is dynamically controllable.

Theorem 3.1. *If N is a solution of the constraint model of dynamic controllability $DC*$, then applying the DC checking algorithm will not tighten any bound of N.*

Proof. First of all, N is consistent otherwise it does not satisfy the shortest path constraints.

For every triangle with one contingent link as shown in Figure 3.1, it must be in one of the three cases (Section 2.2.2.2).

- Follow case where $u_{BC} < 0$: the reduction rule will not tighten any bounds if the triangle is consistent.
- Precede case where $l_{BC} \geq 0$: the reduction rule will tighten link AB to $[u_{AC} - u_{BC}, l_{AC} - l_{BC}]$. According to the constraint model $DC*$, this triangle is restricted by either the linear precede constraints (3.2) or the disjunctive precede constraints (3.3). Both kinds of precede constraints can guarantee that link AB has already satisfied the tightening which means no exact tightening will happen in the precede case.
- Unordered case where $l_{AB} < 0$ and $u_{AB} \geq 0$: either unconditional or conditional reduction is applied. A wait $< C, u_{AC} - u_{BC} >$ is introduced in the unorder case, in the DC checking process. If $u_{AC} - u_{BC} \leq l_{AC}$, the unconditional reduction tightens the lower bound of AB to $u_{AC} - u_{BC}$. Otherwise the general (conditional) unordered reduction will raise the lower bound of AB to l_{AC}. But N satisfies constraints (3.4) and (3.5), thus, neither unconditional nor conditional reduction provides tighter bounds. Therefore, applying reduction rules of local dynamic controllability (regardless of the wait regressions) does not tighten any bound of N.

For the global dynamic controllability (considering wait regressions), the reduction rules of wait regressions are applied to N. The general wait regression reduction is applied for any link DB with upper bound u_{DB}, then a regressed wait on AD is $< C, Wait_{ACB} - u_{DB} >$. The conditional wait regression reduction is applied when $Wait_{ACB} \geq 0$ and DB is a contingent link, a regressed wait on AD is $< C, Wait_{ACB} - l_{DB} >$. Because N satisfies constraints (3.6) and (3.7), the wait regression reduction rules cannot tighten any link. □

The solutions of the constraint model $DC*$ are *minimal*, which means every link in the solutions cannot be tightened by other links any more. However, a dynamically controllable STPU need not to be minimal. For example, in Figure 3.3, the STPU is dynamically controllable, but it violates the precede constraints that link AC has to be within $[0, 4]$. Thus, only dynamically controllable and minimal STPU satisfies the

constraint model $DC*$. To prove the completeness of the constraint model, we show that all dynamically controllable and minimal STPU satisfy the constraint model.

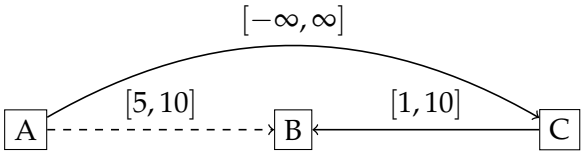


Figure 3.3: An example of a dynamically controllable but not minimal STPU

Theorem 3.2. *Every dynamically controllable and minimal STPU satisfies the dynamic controllability constraints $DC*$. (There is no dynamically controllable and minimal STPU which does not satisfy $DC*$.)*

Proof. Proof by contradiction, suppose there exists a dynamically controllable and minimal STPU N' which does not satisfy the dynamic controllability constraints. Thus, N' violates shortest path, precedence or wait constraints.

- N' does not satisfy a shortest path constraint. N' has one triangle as in Figure 3.2 and the triangle violates a shortest path constraint. This assumption makes the triangle not minimal, which breaks the condition that N' is minimal. Thus, N' satisfies all shortest path constraints.

- N' does not satisfy a precede constraint. N' contains a triangle ABC as shown in Figure 3.1 and the triangle breaks a precede constraint, which means $l_{AB} < u_{AC} - u_{BC}$ (or $u_{AB} > l_{AC} - l_{BC}$). In the precede case, the duration of the contingent link is unknown before executing timepoint B, so no matter how long the duration of contingent link AC is, there exists a schedule $T_B - T_A \in [l_{AB}, u_{AC} - u_{BC})$ (or $T_B - T_A \in (l_{AC} - l_{BC}, u_{AB}]$) such that all temporal constraints in N' are satisfied. If the duration of contingent link AC $T_C - T_A = u_{AC}$ and the scheduled duration of AB is $T_B - T_A = l_{AB} < u_{AC} - u_{BC}$, then the difference between B and C is $T_C - T_B = (T_C - T_A) - (T_B - T_A) > u_{AC} - (u_{AC} - u_{BC}) = u_{BC}$, the duration of BC $T_c - T_B > u_{BC}$, which means it is out of its upper bound. Thus, $l_{AB} \geq u_{AC} - u_{BC}$. It can be proved in the same way that when the duration of contingent link AC is its lower bound the upper bound of AB has to satisfy $u_{AB} \leq l_{AC} - l_{BC}$. Thus, N' does not violate any precede constraints.

- N' does not satisfy a wait constraint, which means that N' has a link AB and $l_{AB} < min(l_{AC}, Wait_{ABC})$. N' is dynamically controllable so that there exists a schedule $T_B - T_A \in [l_{AB}, min(l_{AC}, Wait_{ABC}))$ such that whatever the duration of link AC is, all temporal constraints of N' are satisfied. Then if the duration of contingent link AC is its upper bound, the temporal difference from C to B is $T_C - T_B = (T_C - T_A) - (T_B - T_A) > u_{AC} - min(l_{AC}, Wait_{ABC})$, which means $u_{BC} > u_{AC} - min(l_{AC}, Wait_{ABC})$.

– If $Wait_{ABC}$ is a triangular wait that $Wait_{ABC} = u_{AC} - u_{BC}$ and the duration of link AC is its upper bound u_{AC}, the duration $T_C - T_B > u_{AC} - min(l_{AC}, u_{AC} - u_{BC})$ has to be within $[l_{BC}, u_{BC}]$. If $l_{AC} \leq u_{AC} - u_{BC}$, the duration $T_C - T_B > u_{AC} - l_{AC}$ and $T_C - T_B \leq u_{BC}$, whose linear combination is $l_{AC} > u_{AC} - u_{BC}$ that breaks the condition. Otherwise, $l_{AC} > u_{AC} - u_{BC}$, the duration of BC satisfies $u_{BC} > u_{AC} - (u_{AC} - u_{BC})$ which is an unsatisfiable constraint. Therefore, all links of N' satisfy the triangular wait constraints.

– If $Wait_{ABC}$ is a regressed wait and it is regressed from a triangular wait $Wait_{AD_0C} = u_{AC} - u_{D_0C}$ through a sequence of connected links $E_\Delta = \{D_1D_0, D_2D_1, ..., BD_n\}$, then $Wait_{ABC} = Wait_{AD_0C} - \sum_{e \in E_\Delta} x_e$, where x_e is l_e if link e is contingent, u_e otherwise. N' is dynamically controllable so all nodes in $X_\Delta = \{D_n, ..., D_0\}$ are executed after B. Thus, the execution time T_B has to deal with all uncertain situations of contingent links in E_Δ. When all contingent links in E_Δ have durations as their lower bounds, the upper bound of the sequence of links from B to C is

$$\begin{aligned} &T_C - T_B = (T_C - T_{D_0}) + (T_{D_0} - T_{D_1}) + ... + (T_{D_n} - B) \\ &T_C - T_B \leq u_{D_0C} + \sum_{e \in E_\Delta \cap EU} l_e + \sum_{e \in E_\Delta \cap EC} u_e. \end{aligned} \tag{3.8}$$

If $l_{AC} \leq Wait_{ABC}$, the duration of link BC $T_C - T_B = (T_C - T_A) + (T_A - T_B) > u_{AC} - l_{AC}$, so that

$$l_{AC} > u_{AC} - (T_C - T_B) \geq u_{AC} - (u_{D_0C} + \sum_{e \in E_\Delta} x_e) = Wait_{ABC},$$

which violates the condition. Otherwise, $l_{AC} > Wait_{ABC}$, then $T_C - T_B > u_{AC} - Wait_{ABC}$, which infers that

$$\begin{aligned} &T_C - T_B > u_{AC} - (u_{AC} - u_{D_0C} - \sum_{e \in E_\Delta \cap EU} l_e - \sum_{e \in E_\Delta \cap EC} u_e) \\ &T_C - T_B > u_{D_0C} + \sum_{e \in E_\Delta \cap EU} l_e + \sum_{e \in E_\Delta \cap EC} u_e. \end{aligned} \tag{3.9}$$

Equation (3.8) contradicts (3.9). Therefore, all regressed waits satisfy the wait bound constraints.

Thus, the links in N' satisfy triangular and regression wait constraints.

In conclusion, every dynamically controllable and minimal STPU satisfies the constraints of dynamic controllability. □

3.2.2 Reducing the Size of the Model

The model formulated above has up to $O(n^3)$ constraints and $O(|V_U|n^2)$ variables, where $n = |V|$ is the number of time points. Wah and Xin [2004, 2007] proposed

several rules for eliminating redundant constraints and variables from the model. Although there is, in principle, a requirement link for every pair of time points, not all of these must be represented with decision variables. For example, if, in Figure 3.1, $U_{BC} < 0$ so that B must follow C, only the shortest path constraints apply to l_{AB} and u_{AB}, in this triangle. If there are no other constraints on e_{AB} these can always be satisfied, in which case it is not necessary to include them. Any link with bounds constrained in the input STPU must be represented (unless fixed to a constant), as must any link whose bounds can potentially be tightened by precede or wait constraints. Among the remaining implicit links, which are subject only to shortest path constraints, a sufficient set is found by a triangulation of the network.

The rules to remove redundant constraints and variables by Wah and Xin [2004] are based on two assumptions: (1) before formulating the constraint model, the loose bounds satisfy dynamic controllability constraints and (2) the constraints $L_r \leq l_r \leq u_r \leq U_r$ and $L_c = l_c = u_c = U_c$ always hold for all requirement and contingent links, respectively. However, these assumptions do not always hold for every application. For example, in the relaxation of over-constrained problem [Yu et al., 2014], the original loose bounds are not dynamically controllable and in the max delay measure, the upper bounds of contingent links are not equal to the loose bounds. Although giving infinite relaxations to the relaxable links can generate the dynamically controllable loose bounds, the infinite bounds on requirement links $[-\infty, \infty]$ can hardly help to reduce constraints. So the following rules may not improve much in some applications.

Furthermore, the reduced model introduced in this subsection is slightly different regarding wait constraints from Wah and Xin's model. Our model considers fewer wait constraints based on the following intuitions.

- If wait bound constraints influence the lower bound of a link, it is the same as the precede constraint on the lower bound. Because the wait is $w_{ABC} \geq u_{AC} - u_{BC}$, if it influence the lower bound $l_{AB} \geq w_{ABC} \geq u_{AC} - u_{BC}$ and the precede constraint on the lower bound is $l_{AB} \geq u_{AC} - u_{BC}$. So we can treat some wait constraints as precede constraints on lower bounds only, which means precede constraints are more strict than wait bounds constraints because they also restrict upper bounds. If the bounds of a triangle, as shown in Figure 3.1, satisfy the precede constraints, no wait constraints are needed, and the triangle is strongly controllable, since no matter the duration of AC is, all schedules of B cannot break any constraint.

- No wait regression is needed, if a wait is positive and smaller than the lower bound of the contingent link. Because the wait may tighten the lower bound of the link where the wait exists, its wait regressions can be replaced by precede constraints if it regresses through a contingent link or shortest path constraints if it regresses through a requirement link. Thus, no wait regresses from this kind of waits.

- Wait bounds constraints are not necessary for all wait variables. This means

that if the wait between two nodes will always satisfy its wait bounds constraints, no new link will be added if there is no link between the two nodes. This may help to build fewer triangles when formulating shortest path constraints.

Therefore, for each triangle, if we cannot guarantee it will be in the precede case (based on loose bounds), wait constraints are considered; Wait regressions are added only if the wait is greater than the lower bound of the contingent link it waits for; Shortest path constraints are always added except when the linear precede constraints are used.

3.2.2.1 Reducing Redundant Shortest Path Constraints

For a triangle, although shortest path constraints are weeker than precede and wait constraints, they are the fundamental bound propagations to keep the network consistent. Before introducing the ways to reduce redundant precedence and wait constraints, we have to ensure that the whole network satisfies the shortest path constraints.

The naive formulation contains $O(N^3)$ shortest path constraints, since it adds a new link to every pair of nodes that are not connected in the original problem. Wah and Xin [2004] introduced a method that adds new links for pairs of nodes that are neighbors by connecting to the same node. It works by selecting one node at a time and formulating shortest path constraints for all current neighbors. After formulating shortest path constraints for all triangles with the common node, the algorithm deletes the node. The iteration repeats until all nodes are deleted. A heuristic function

$$h_A = \frac{2\|\{e_{ij}|x_i, x_j \in N_A, e_{ij} \in N\}\|}{\|N_A\|(\|N_A\| - 1)} \tag{3.10}$$

decides the order to pick up nodes, where N_A is the set nodes connected to node A and $\|S\|$ means the size of set S. Every iteration, the node with largest h_A will be selected, triangles containing the nodes will be formulated and the node will be deleted from the graph then. We adopt this method in our implementation.

3.2.2.2 Reducing Redundant Precedence Constraints

The precede constraints that can be removed are implied by other constraints that cannot be removed. Precede constraints are the most strict constraints for a triangle, so it cannot be implied by a wait constraint or a shortest path constraint only. Thus, it may be implied only by other precede constraints caused by altering loose bounds.

Linear precede constraints (Equation 3.2) aim to restrict triangles that are in precede case and disjunctive linear precede constraints (Equation 3.3) aim to restrict triangles that may be in precede case. For each contingent link AC, we can classify the rest of the nodes into three subsets as in the following formula according to [Wah

and Xin, 2004].

$$\begin{cases} \{B \| L_{BC} \geq 0\} & \text{(precede set)} \\ \{B \| U_{BC} < 0\} & \text{(post set)} \\ \text{otherwise} & \text{(unordered set)} \end{cases} \tag{3.11}$$

In the naive formulation, we add linear precede constraints for all nodes in precede set and disjunctive linear precede constraints for all nodes in the unordered set. The variables involved in those constraints that have not been added will be added.

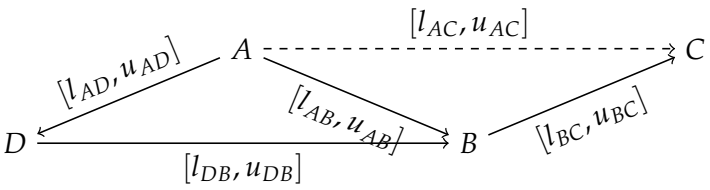


Figure 3.4: Reduce redundant precede constraints

However, the constraints for nodes in precede set can be reduced. For instance, as shown in Figure 3.4, if both B and D are in the precede set, precede constraints for triangle ADC and variables for the new link DC are added in the naive formulation following equation (3.12), but they are redundant.

$$\begin{array}{rcl} l_{AD} & \geq & u_{AC} - u_{DC} \\ u_{AD} & \leq & l_{AC} - l_{DC} \end{array} \tag{3.12}$$

Since link DC is an added link, its bounds are constrained by shortest path constraints $l_{DC} \geq l_{DB} + l_{BC}$ and $u_{DC} \leq u_{DB} + u_{BC}$. The equations above can be transformed into the following equations without variables of link DC,

$$\begin{array}{rcccl} l_{AD} & \geq & u_{AC} - u_{DC} & \geq & u_{AC} - (u_{DB} + u_{BC}) \\ u_{AD} & \leq & l_{AC} - l_{DC} & \leq & l_{AC} - (l_{DB} + l_{BC}) \end{array}$$

which are implied by the precede constraints in triangle ABC and shortest path constraints in ADB, if the link DC is not added by other constraints formulation.

$$\begin{array}{rcccl} l_{AD} & \geq & l_{AB} - u_{DB} & \geq & (u_{AC} - u_{BC}) - u_{DB} \\ u_{AD} & \leq & u_{AB} - l_{DB} & \leq & (l_{AB} - l_{AC}) - l_{BC} \end{array}$$

The node B is called a *guard node* since it guards the precede constraints for the other nodes in the precede set that connected to AC through it. We can use Depth First Search (DFS) or Breadth First Search (BFS) starting from node C to collect guard nodes. The nodes are in precede set but not guard nodes can be ignored when formulating precede constraints. The detailed GuardDFS is illustrated in Algorithm 1, which returns if the current node has been visited or is a guard node (Line 1), or marks the current node as a guard node and returns if the current node is in the

precede set (Line 5), otherwise calls recursively (Line 9) on nodes connected to the current node.

Unfortunately, nodes in the unordered set do not have guard nodes because the loose bounds cannot help decide which ones are in precede case. Therefore, all precede constraints are necessary for nodes in the unordered set.

The process of adding precede constraints for contingent link AC is shown in Algorithm 2. Algorithm 2 is the general process. Line 3 – 11 describe the initial classifications of all nodes, -2 for the precede set, 1 for the post set and 0 for the unorder set. Line 12 – 16 describe the method to mark guard nodes, which calls GuardDFS from node C. After marking guard nodes, Line 17 to the end of Algorithm 2 add new links and linear or disjunctive linear precede constraints for nodes in the precede and unorder set respectively.

Algorithm 1: The algorithm finding guard nodes

Algorithm: *GuardDFS* (x, v, *class*, E)
Input: The current node x, a vector of visited mark v, the classification of nodes *class* and the edges E
Output: The updated v and *class*

```
1  if v[x] == True or class[x] == guard then
2  |  return ;                          // x has been visited, return
3  endif
4  v[x] = True;
5  if class[x] == prec then
6  |  class[x] = guard;                  // mark x as a guard node
7  |  return;
8  endif
9  for xy ∈ E do
10 |  GuardDFS( y, v, class, E);         // search connected nodes
11 end
```

3.2.2.3 Reducing Redundant Wait Constraints

Similar to the method to reduce redundant precede constraints, the way to reduce wait constraints classifies nodes into pre-wait, post-wait and unordered-wait subsets, and then removes redundant constraints. But wait constraints are more complex because regressed waits may also influence the bounds, so we will discuss triangular waits in different subsets first, and regressed waits will be discussed after that.

We adopt the classification from Wah and Xin [2004] but slightly modify the reduction rules they use. For each contingent link AC, we classify the nodes in the following way:

$$\begin{cases} \{B \| U_{AB} \leq L_{AC}\} & \text{(pre-wait set)} \\ \{B \| L_{AB} \geq L_{AC}\} & \text{(post-wait set)} \\ \text{otherwise} & \text{(unordered-wait set).} \end{cases} \tag{3.13}$$

In the pre-wait set, it is redundant to formulate triangular wait constraints and wait regression constraints starting from the nodes. If we use shortest path constraint

Algorithm 2: The algorithm formulating precede constraints of dynamic controllability

Algorithm: $addPrecedeConstrs(N, A, C, M)$
Input: An STPU $N = \langle X, E = R \cup C \rangle$, a contingent link AC and the constraint model M.
Output: The updated STPU N, the updated constraint model M. Return True if new links are added, otherwise return False.

```
1  update = False;
2  class = [] ;                                   // the classification of nodes
3  for x ∈ V do
4      if L_xC ≥ 0 then
5          class[x] = prec ;                                  // precede set
6      else if U_xC < 0 then
7          class[x] = post ;                                  // post set
8      else
9          class[x] = unorder ;                               // unorder set
10     endif
11 end
12 for x ∈ V do
13     v[x] = False;
14 end
15 v[A] = True;
16 GuardDFS(C, v, class, E);
17 for x ∈ V do
       /* add new links                                                  */
18     if class[x] == guard||class[x] == unorder then
19         if Ax ∉ E then
20             update = True;
21             N.add(Ax);
22         endif
23         if xC ∉ E then
24             update = True;
25             N.add(xC);
26         endif
27     endif
       /* add new constraints                                            */
28     if class[x] == guard then
29         M.addLPrecede(AC, x);                          // Equation (3.2)
30     else if class[x] == unorder then
31         M.addDlPrecede(AC, x);                         // Equation (3.3)
32     endif
33 end
34 return update;
```

$u_{AB} + l_{BC} \geq l_{AC}$, it can be deducted that $l_{BC} \geq 0$, which makes B in precede case for contingent link AC. Thus, precede constraints will be considered that are more strict than wait constraints. Therefore, triangular waits are redundant for nodes in the pre-wait set. Furthermore, the regressed waits for contingent link AC will be no larger than the lower bound of AC, which means if the wait is positive, constraint

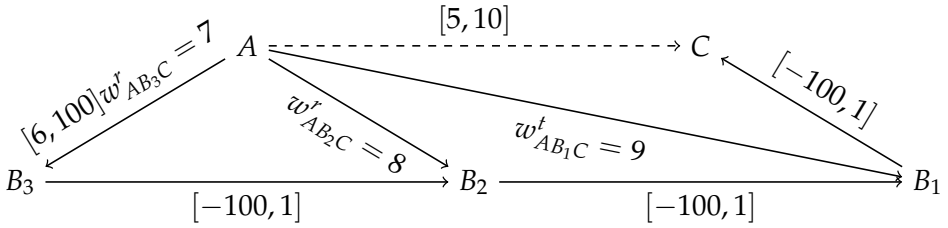


Figure 3.5: Reduce redundant triangular wait constraints I

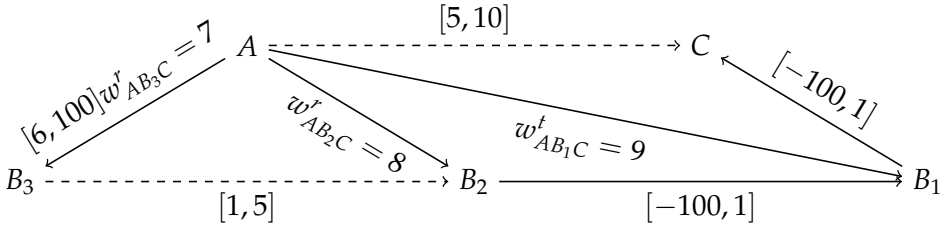


Figure 3.6: Reduce redundant triangular wait constraints II

$l_{AB} \geq w_{AB}$ holds. Therefore, the wait regressions from a wait in the pre-wait set can be replaced by precede constraints if it regresses through a contingent link, or shortest path constraints if it regresses through a requirement link.

The unordered-wait set is where the waits may tighten the bounds. Triangular waits and wait bounds are necessary.

In the post-wait set, wait bound constraints are redundant. Because among regression waits and triangular wait, only the largest one matters. The largest wait on AB is no less than its triangular wait $w_{ABC} \geq u_{AC} - u_{BC}$, plus shortest path constraint $u_{AC} - u_{BC} \geq l_{AB}$ infers that $w_{ABC} \geq l_{AB} \geq l_{AC}$. In conclusion, waits in the post-wait set cannot tighten the bounds. Therefore, we only need to consider their regression waits.

Some nodes in the post-wait set do not have triangular waits that can regress. If $U_{BC} < 0$, which means the time to schedule B will always be after the observation of C, this triangle is in the follow-case that does not have a wait. For the nodes in the post-wait set that are not in the follow case, redundant triangular waits still exist. For instance, in Figure 3.5, B_1, B_2 and B_3 are all in the post-wait set, the regression waits of w_{AB_1C} through requirement links will never tighten their bounds. Additionally, the regressions can be replaced by triangular wait in triangle AB_3C and shortest path constraints of triangle B_3B_2C and B_2B_1C. However, if $B_3 \to B_2$ is a contingent link in Figure 3.6, the regression from w_{AB_2C} to w_{AB_3C} is necessary because it cannot be replaced by other constraints (although B_3B_2A is in precede case, w_{AB_2C} has not influenced l_{AB_2}, which cannot pass to AB_3). Therefore, in the post-wait set, we formulate triangular waits for nodes D and B that $U_{BC} \geq 0$ and DB a contingent link.

Regression waits describe the constraints of global dynamic controllability. A wait regression consists of a source wait, a regressed wait and the link between the end nodes of the links where the two waits constrain. The source waits of regression waits are from the nodes in the unordered-wait and post-wait sets because regression waits from the nodes the pre-wait set can be replaced by precede and shortest path constraints. Furthermore, the waits regressing through two requirement links continuously can be replaced by one regression and a shortest path constraint, so the source waits are from the start nodes of contingent links. The regressed waits fall in any sets. The regressed waits in the post-wait set cannot tighten their lower bounds, thus only regressed waits that may regress again will be considered in the post-wait set, which are the regressed waits to the end of contingent links in the post-wait set. The regressed waits in the pre-wait and unordered-wait sets cannot be reduced because they can influence their lower bounds. Based on the reduction rules, waits regress through contingent links or requirement links with positive upper bounds, if the waits are greater than the lower bounds of the contingent links they wait. Wait regressions through contingent links are formulated if the end nodes of the contingent links have triangular or regressed waits. Wait regression through requirement links are formulated in the following cases.

- Wait regressions from nodes in the post-wait or unordered-wait set have to come from start nodes of contingent links, if they regress through requirement links. Otherwise, the wait is regressed via two connected requirement links, which can be replaced by a wait regression and a shortest path constraint.
- Wait regressions to nodes in the post-wait set regress to the end nodes of contingent links because the regressed waits will be regressed again through a contingent link. Otherwise, the regressions can be implied, too.
- Wait regressions to the nodes in the unordered-wait and pre-wait sets can regress to any nodes, since the regressed waits may tighten the lower bounds if they are smaller than the lower bound of the contingent link they wait for. But only waits of the start nodes of contingent links in the unordered-wait set can be regressed again.
- Only guard nodes in the pre-wait set will be considered to formulate wait regressions.

Wait bounds will be added to waits of nodes in both pre-wait and unordered-wait sets.

The summary of how to add wait constraints are shown in Algorithms 3 and 4. First, the initial classification of all nodes are given from Line 3 to 13. Then the algorithm finds the guard nodes in the pre-wait set with Line 15–19. The method *GuardDFS* used here is the same as the one for finding the guard nodes when adding precede constraints. After dividing all the nodes, Line 20 to 32 of Algorithm 3 add triangular waits and new links for (1) the guard nodes in the pre-wait set, (2) all the nodes in the unordered-wait set and (3) the end nodes of contingent links in the

post-wait set. The last line of algorithm 3 will raise a function call of algorithm 4 which adds regression waits and wait bounds.

In Algorithm 4, regression waits via contingent links are introduced for contingent links first (Line 3). Since regression waits cannot be through two consecutive requirement links, regressed waits via requirement links are added from the waits just regressed from contingent links (Line 6). Line 11 to the end of the algorithm add wait bounds constraints.

3.2.2.4 Summary

We have to claim that the overall process to formulate the reduced model is not independent for precede, wait and shortest path constraints, since adding new links will result in different guard nodes sets. Therefore, we add precede and wait constraints and associated variables first and add shortest path constraints and associated variables then. The new links added for formulating shortest path constraints may introduce more strict constraints to precede and wait constraints. Therefore, the overall process is iterative, we need to iterate all steps until no new link is added. The process is shown in Algorithm 5. Function *addSPC* adds the shortest path constraints. These are formulated using the method explained in section 3.2.2.1.

3.2.3 Formulation as a Mixed Integer Programming (MIP) Model

The disjunctions in constraints (3.3) and (3.5) mean the model is not a linear program. Wah and Xin [2004] used non-linear constraints to encode the disjunctions (as explained in the next section) and tackled it with the non-linear programming solver SNOPT. As an alternative, we formulate a mixed-integer linear programming (MIP) formulation, where disjunctions are encoded using binary (0/1) variables. Although MIP is an NP-hard problem, MIP solvers such as CPLEX or Gurobi are often very efficient in practice, and, in particular, typically more efficient than non-linear solvers. Experiment results across all application problems confirm this.

The wait bound constraint (3.5) can be formulated as

$$\text{if } \alpha > 0 \text{ then } \beta \geq 0 \text{ else } \gamma \geq 0$$

where $\alpha = w_{ABX} - l_{AX}$, $\beta = l_{AB} - l_{AX}$ and $\gamma = l_{AB} - w_{ABX}$ are all linear expressions. The disjunction can be replaced by the following linear constraints

$$\begin{aligned} \alpha - xU_\alpha &\leq 0 && \text{(3.14a)} \\ \alpha - (1-x)(L_\alpha - 1) &> 0 && \text{(3.14b)} \\ \beta - (1-x)L_\beta &\geq 0 && \text{(3.14c)} \\ \gamma - xL_\gamma &\geq 0 && \text{(3.14d)} \end{aligned}$$

where $x \in \{0,1\}$ is a binary variable, L_α, L_β and L_γ are constant lower bounds on α, β and γ, respectively, and U_α is a constant upper bound on α. This forces $\alpha > 0$ and $\beta \geq 0$ when $x = 1$, and $\alpha \leq 0$ and $\gamma \geq 0$ when $x = 0$. In the wait bound constraint,

Algorithm 3: The algorithm formulating wait constraints of dynamic controllability

Algorithm: $addWaitConstrs(N, A, C, M)$
Input: An STPU $N = \langle V = V_E \cup V_U, E = EC \cup EU \rangle$, a contingent link AC and the constraint model M.
Output: The updated STPU N, the updated constraint model M. Return True if new links are added, otherwise return False.

```
1  update = False;
2  class = [] ;                                   // the classification of nodes
3  for x ∈ V do
4      if U_Ax ≤ L_AC then
5          class[x] = prec ;                                  // pre-wait set
6      else if L_Ax ≥ L_AC then
7          class[x] = post − II ;                             // post-wait set
8          if U_xC ≥ 0 and x ∈ V_U then
9              class[x] = post − I;
10         endif
11     else
12         class[x] = unorder ;                          // unordered-wait set
13     endif
14 end
15 for x ∈ V do
16     v[x] = False;
17 end
18 v[A] = True;
19 GuardDFS(C, v, class, E);
20 for x ∈ V do
21     if class[x] == unorder or (class[x] == post − I and x ∈ V_U) then
           /* add new links                                                  */
22         if Ax ∉ E then
23             update = True;
24             N.add(Ax);
25         endif
26         if xC ∉ E then
27             update = True;
28             N.add(xC);
29         endif
           /* add triangular wait constraints                                */
30         M.addTriWait(AC, x);                          // Equation (3.4)
31     endif
32 end
33 addRWait(N, AC, class, M);                    // add wait regressions
34 return update;
```

where $\alpha = w_{ABX} - l_{AX}$, we can choose $U_\alpha = U_{AB} - L_{AX}$, because U_{AB} is an upper bound on w_{ABX} ($w_{ABX} \leq u_{AB} \leq U_{AB}$) and L_{AX} is a lower bound on l_{AX}. The wait w_{ABX} is lower-bounded by the maximum of all wait constraints – triangular and regressed – on e_{AB}. The triangular wait lower bound is $w^t_{ABX} = u_{AX} - u_{BX}$, and from

Algorithm 4: The algorithm formulating regression wait constraints of dynamic controllability

Input: An STPU $N = \langle V = V_E \cup V_U, E = EC \cup EU \rangle$, a contingent link AC the classification of all nodes *class* and the constraint model M.
Output: The updated constraint model M.
Algorithm: $addRWait(N, AC, class, M)$

for $yx \in EU$ **do**
  if $class[x] == unorder\ or\ class[x] == post - I$ **then**
    $M.addRegWaitC(AC, x, y)$; // add $w_{AyC} \geq w_{AxC} - l^c_{yx}$
    for $z \in V$ **do**
      if $(class[z] == guard\ or\ class[z] == unorder\ or\ (class[z] == post - I\ and\ z \in V_U))\ and\ U_{zy} \geq 0$ **then**
        $M.addRegWaitR(AC, y, z)$; // add $w_{AzC} \geq w_{AyC} - u^r_{zy}$
      endif
    end
  endif
end
for $x \in V$ **do**
  if $class[x] == unorder\ or\ class[x] == guard$ **then**
    $M.addWaitBounds(Ax)$; // Equation (3.5)
  endif
end

Algorithm 5: The general algorithm formulating dynamic controllability constraints

Input: An STPU $N = \langle V, E = EC \cup EU \rangle$.
Output: A dynamic control constraint model M
Initialization:
$M.variables = \emptyset$; // initialize an empty model
$M.constraints = \emptyset$;
update = True;
Algorithm: $addDCConstrs(N)$
while $update == True$ **do**
  $update = False$;
  for $e \in EU$ **do**
    $update| = addPrecedeConstrs(N, e.start, e.end, M)$;
    $update| = addWaitConstrs(N, e.start, e.end, M)$;
  end
  while $x = selectNext(N)$ **do**
    $update| = addSPC(x, N, M)$;
    $N.delete(x)$;
  end
end
`return` M

the shortest path constraint $l_{AB} + u_{BX} \leq u_{AX}$ we have $w^t_{ABX} \geq l_{AB}$. Thus, we can choose $L_\alpha = L_{AB} - U_{AX}$. For the lower bounds on $\beta = l_{AB} - l_{AX}$ and $\gamma = l_{AB} - w_{ABX}$

we have $L_\beta = L_{AB} - U_{AX}$ and $L_\gamma = L_{AB} - U_{AB}$, respectively.

The precede constraint (3.3) and regressed wait bound (3.6) are similar, except they have conditions only in one of the two cases (either "then" or "else"). Where one side of a disjunction consists of (a conjunction of) several linear constraints, as in (3.3), it is only necessary to add a constraint like (3.14c) for each conjunct, all using the same binary variable.

The wait regression constraint (3.6) can be strengthened to

$$(w_{ABX} \leq l_{AX}) \quad \vee \quad (w_{ADX} \geq w_{ABX} - l_{DB}) \tag{3.15}$$

The advantage of this is that one disjunct, $w_{ABX} \leq l_{AX}$, is the same as the branching condition in (3.5), so both constraints can be captured with one binary variable.

Constraint (3.15) is valid because regressing a wait w_{ABX} through a contingent link e_{DB}, via (3.6), is redundant if w_{ABX} is not greater than the lower bound of the contingent link e_{AX} that causes the wait. If $w_{ABX} \leq l_{AX}$, the wait bound constraint (3.5) implies $l_{AB} \geq w_{ABX} \geq 0$. Hence, the triangle DAB is in the precede case and $l_{AD} = -u_{DA} = -(l_{DB} - l_{AB}) = w_{ABX} - l_{DB}$, which implies the wait regression constraint (3.6).

3.2.4 Formulation as a Non-linear Programming (NLP) Model

The non-linear model formulated by Wah and Xin [2004] uses quadratic constraints and terms in the objective function to encode disjunctions. The precede constraint (3.3) is formulated as follows:

$$l_{BC}(l_{AB} - u_{AC} + u_{BC}) \geq 0 \tag{3.15a}$$
$$l_{BC}(u_{AB} - l_{AC} + l_{BC}) \leq 0 \tag{3.15b}$$

For each wait bound constraint (3.5), they introduce an auxiliary variable β and the following constraints:

$$(l_{AX} - w_{ABX})(l_{AB} - w_{ABX}) \geq 0 \tag{3.16a}$$
$$\beta \geq 0 \tag{3.16b}$$
$$\beta \geq w_{ABX} - l_{AX} \tag{3.16c}$$
$$\beta(l_{AB} - l_{AX}) \geq 0 \tag{3.16d}$$

Furthermore, a quadratic term $\beta(\beta - (w_{ABX} - l_{AX}))$ is added to the objective function. (This term must be minimised; if the problem is one of maximisation, its negative is used.) Its purpose is to ensure that β is 0 when $w_{ABX} \leq l_{AX}$ and otherwise equal to the difference $w_{ABX} - l_{AX}$. For this to work, the penalty incurred by a non-zero value of this must term outweigh the actual objective function. Note also that there is a situation in which this formulation fails to impose the lower bound on l_{AB}, since if $l_{AX} = w_{ABX}$, constraint (3.16a) is satisfied even if $l_{AB} \not\geq w_{ABX}$, and (3.16b)–(3.16d) are satisfied by setting $\beta = 0$. A similar encoding is used for the wait

regression constraint (3.15).

3.2.5 Conflict-Directed Relaxation with Uncertainty

Finally, we consider the Conflict-Directed Relaxation with Uncertainty (CDRU) algorithm Yu et al. [2014]. CDRU takes an STPU that is *not* dynamically controllable and finds a least-cost relaxation of the temporal constraints that makes it controllable. Relaxing an STPU means tightening contingent links (reducing uncertainty) and/or loosening requirement links. The cost of a relaxation is a function of the change to each link, and some links may be excluded from change.

Relaxing over-constrained STPUs is one application of optimising bounds subject to dynamic controllability (cf. Section 3.4.1). In other applications, the initial STPU may already be dynamically controllable and the aim is to improve the value of an objective function by increasing uncertainty or tightening requirements while maintaining controllability. CDRU can be adapted to such problems by taking a dual approach: initial link bounds are chosen to maximise the objective, disregarding controllability, and the resulting (typically uncontrollable) STPU is then gradually relaxed to a feasible solution. An application is able to solve by CDRU iff it can be tightened into a not controllable STPU to start with. For example, in the application of minimising flexibility (cf. Section 3.4), the original problem is a dynamically controllable STPU, which is tricky to be solved by CDRU. Last but not least, the solutions returned by CDRU are not minimal.

CDRU draws on conflict-directed A* [Williams and Ragno, 2002], extending conflict learning and resolution to continuous variables [Yu and Williams, 2013] and contingent constraints. The algorithm explores candidate relaxations in best-first order by interleaving two steps:

- First, it runs a dynamic controllability checking algorithm on the current best candidate STPU. If it is not dynamically controllable, the check returns an unsatisfied disjunction of linear constraints over subsets of link bounds. This is called a conflict: at least one constraint in the set must be satisfied to make the STPU controllable. The conflict is recorded as a new constraint that any solution must satisfy.

- Given a set of conflicts, the algorithm computes least-cost relaxations that eliminate them. A relaxation may loosen the bounds of requirement links, or tighten the bounds of contingent links. Since conflicts can be disjunctive, a conflict may generate several new candidates.

To apply CDRU to optimisation problems other than linear-cost relaxations, we have to replace the objective function and redefine the initial candidate accordingly. For example, in the maximum delay problem (see Section 3.4), the objective is to maximise the minimum width of contingent link intervals. Thus, we need a max-min function: $\max(\min_{e_{ij} \in EU}(u_{ij} - l_{ij}))$. Finding the least-cost resolver of a single conflict disjunct (linear constraint) under this cost function can still be formulated as a linear

program, and solved by a fast linear optimiser. The revised version of CDRU is shown as Algorithm 6. The input is an STPU in which the upper bounds of all contingent links are set to a very large number[1]. A candidate is a pair $\langle UB, C_r \rangle$, where UB is a set of tightenings to the upper bounds of contingent links and C_r the set of conflicts resolved by this candidate. For the initial candidate both sets are empty.

Algorithm 6: The CDRU algorithm adapted for the maximum delay problem.

Algorithm: CDRU(N)
Input: An STPU $N = \langle V, E = EC \cup EU \rangle$.
Output: A set of tightened contingent link upper bounds $\{u_{ij} | e_{ij} \in EU\}$ making N dynamically controllable.

```
1  Cand ← ⟨UB, C_r⟩; the first candidate;
2  Q ← {Cand}; a priority queue of candidates;
3  Cft ← ∅; the set of known conflicts;
4  while Q ≠ ∅ do
5  |  Cand ←Dequeue(Q);
6  |  currCft ←UnresolvedConflicts(Cand, Cft);
7  |  if currCft == null then
8  |  |  newCft ← DynamicControllable?(Cand)
9  |  |  if newCft == null then
10 |  |  |  return Cand;
11 |  |  else
12 |  |  |  Cft ← Cft ∪ {newCft};
13 |  |  |  Q ← Q ∪ {Cand};
14 |  |  endif
15 |  else
16 |  |  Q ← Q∪ExpandOnConflict{Cand, currCft}
17 |  endif
18 end
19 return null;
```

In Figure 3.7(a), the STPU three contingent and three requirement links. The upper bounds of the contingent links are initialised to 10^7. This STPU is not dynamically controllable since the upper bound of link BF is less than the sum of the upper bounds of CD and EF. This generates a conflict involving five links: $u_{BF} - l_{DE} - u_{CD} - u_{EF} - l_{BC} \geq 0$. The least-cost resolver of this conflict tightens u_{CD} and u_{EF}, and generates the new best candidate shown in Figure 3.7(b). However, this candidate is still not controllable. Checking returns the conflict $u_{BF} - u_{EF} - u_{CD} - l_{BC} - u_{AB} + l_{AB} - l_{DE} \geq 0 \vee u_{BF} - u_{EF} - u_{CD} - l_{DE} \geq 0$. This conflict is a disjunction of two linear inequalities, meaning it can be resolved in two ways. This is the property of dynamic controllability: we can either apply relaxations to make the "negative cycles" in the graph positive, or apply relaxations to shift the order of some constraints such that the cycle itself is eliminated. Only one of the linear inequalities needs to be satisfied to resolve this conflict. CDRU finds two alternative

[1] Due to numerical issues, we do not use $+\infty$

sets of relaxations, leading to the two new candidates shown in Figure 3.7(c) and (d). It first considers candidate (c), since it allows a width of 4 for all contingent links, while candidate (d) allows only 3. Since candidate (c) is also controllable, it is returned as the preferred solution to the input STPU.

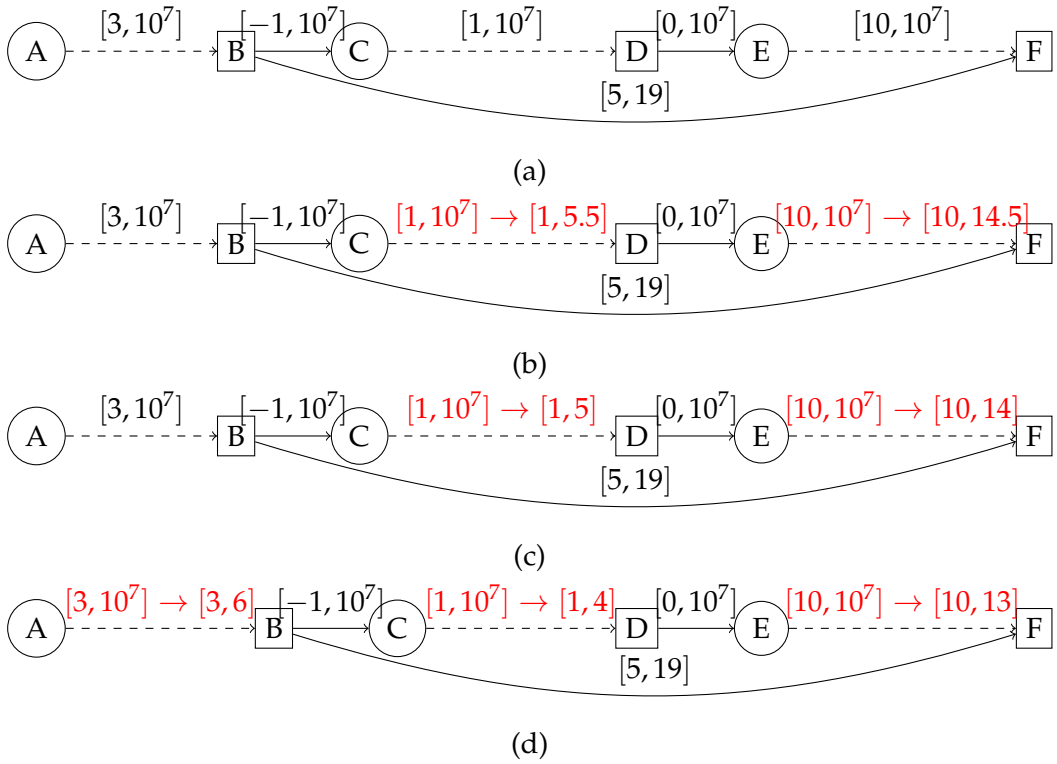


Figure 3.7: (a) Example of an uncontrollable STPU (upper bounds on contingent links are too large); (b–d) first, second and third relaxation candidate.

3.3 Constraint Model of Strong Controllability

In this section, we describe the constraint model of strong controllability. The strong controllability checking algorithm Vidal and Ghallab [1996] is polynomial. The reduction rules in the algorithm consider worst cases of the uncertain situations and apply propagation algorithms (e.g. Floyd-Warshall or PC-2) to the resulting STN. Based on the reduction rules, Fang et al. [2014] describe an optimisation model for chance-constrained probabilistic STP under constraints of strong controllability. However, the propagations in the algorithm are based on time points which are different from our general optimisation model that treats bounds of links as variables. Therefore, we introduce the reduction rules of strong controllability on links and introduce a new constraint model of strong controllability based on them.

3.3.1 Strong Controllability Reduction Rules

Strong controllability can be regarded as a special class of dynamic controllability because a strongly controllable STPU implies its dynamic controllability.

The reduction rules of strong controllability can be shown in the same way as those of dynamic controllability (see Section 2.2) but more strict. To be more specific, strong controllability reduction rules are the same as dynamic controllability in precede case because the "universal" solution has to be made without observations of contingent links. For instance, in the triangular example in Figure 3.1, if the duration of the contingent link AC is its lower bound L_{AC} (or upper bound U_{AC}), whatever L_{BC} and U_{BC} are, $L_{AB} \leftarrow \max(L_{AB}, U_{AC} - U_{BC})$ and $U_{AB} \leftarrow \min(U_{AB}, L_{AC} - L_{BC})$.

3.3.2 Constraint Model of Strong Controllability

The constraint model of strong controllability is a Linear Program. Constraints are formulated over each triangle the same as the constraint model of dynamic controllability.

For all triangles with one contingent link, linear precedence constraints (Equation 3.2') are introduced. For other triangles, only the shortest-path constraints (Equation 3.1) are used.

The correctness of the constraint model can be proved by illustrating that all solutions satisfy the strong controllability reduction rules.

3.3.3 Reducing the Size of the Model

Checking strong controllability can be done by a conjunction of local satisfiability checks [Vidal and Fargier, 1999]. The constraint model of strong controllability only needs to formulate constraints for local satisfiability as well.

For precede constraints, the strong controllability model only formulates constraints for requirement links that connect to the end nodes of contingent links. New links will be added if they do not exist when formulating triangles. This process guarantees the local satisfiability of strong controllability.

After formulating precedence constraints, we can formulate shortest path constraints following the same process we use in formulating the constraint model of dynamic controllability.

The process to add constraints of strong controllability is straightforward, so we skip its pseudo code.

3.4 Applications

Next, we review different problems from the literature that can be formulated as the optimisation over an STPU with dynamic controllability constraints, and compare the effectiveness of different methods of solving them.

All experiments were run on 3.1GHz AMD cores with 64Gb memory.

3.4.1 Relaxing Over-Constrained Problems

An STPU that is not dynamically controllable often arises in planning and scheduling because users' goals (requirements) are too ambitious or the desired robustness to uncertainty (width of contingent links) is too great. In this situation, dynamic controllability can be restored by relaxing the problem: widening requirement links and/or tightening contingent links [Yu et al., 2014].

Let $\hat{l}_{ij}$ and $\hat{u}_{ij}$ denote the original (desired) bounds on link e_{ij}. The relaxation problem can be formulated as

$$\begin{aligned}
\min \quad & \sum_{e_{ij} \in E} f_{ij}(\delta^l_{ij}, \delta^u_{ij}) \\
\text{s.t.} \quad & l_{ij} = \hat{l_{ij}} - \delta^l_{ij} + \tau^l_{ij} \leq u_{ij} = \hat{uu_{ij}} + \delta^u_{ij} - \tau^u_{ij} \qquad e_{ij} \in EC \\
& l_{ij} = \hat{l_{ij}} + \delta^l_{ij} \leq u_{ij} = \hat{uu_{ij}} - \delta^u_{ij} \qquad e_{ij} \in EU \\
& \delta^l_{ij}, \delta^u_{ij}, \tau^l_{ij}, \tau^u_{ij} \geq 0 \\
& \text{dynamic controllability (3.1)–(3.7)}
\end{aligned}$$

where $f_{ij}(\delta^l, \delta^u)$ encodes the relative cost relaxing the lower and upper bounds on link e_{ij} by δ^l and δ^u, respectively. The dynamic controllability constraints enforce not only that the network is dynamically controllable, but also that bounds are minimal (i.e., the tightest implied) on each requirement link. Because of this, we must allow also for requirement links to be tightened, without affecting the objective function. This is why deviations from the target value are split into two non-negative variables, e.g., δ^l_{ij} and τ^l_{ij} for the lower bound, and only the relaxation part appears in the objective. (Note that minimal bounds are computed also as part of the reductions made by the DC checking algorithm, but a network does not have to be minimal to be dynamically controllable.)

For contingent links, we can set the constants $L_{ij} = \hat{l_{ij}}$ and $U_{ij} = \hat{uu_{ij}}$, since their bounds can only shrink. For requirement links there are no given limits in this problem.

3.4.1.1 Comparison of Solvers

We compare the conflict-directed relaxation procedure (CDRU), the MIP model solved with Gurobi[2] and the non-linear model solved with SNOPT[3] on 2400 relaxation test cases used by Yu et al. [2014]. The STPUs have between 14 and 2000 nodes, and a number of contingent and requirement links approximately linear in the number of nodes. The objective function is a linear function of the amount of relaxation; it is symmetric w.r.t. relaxation of lower and upper bounds (i.e., $f_{ij}(\delta^l_{ij}, \delta^u_{ij}) = c_{ij}(\delta^l_{ij} + \delta^u_{ij})$).

Not surprisingly, the CDRU is the fastest on this problem and scales much better than both the MIP and non-linear solver, as shown in Figure 3.8(a). The non-linear solver does not guarantee solution optimality: of the solutions it finds on the relax-

[2] http://www.gurobi.com/

[3] http://ccom.ucsd.edu/~optimizers/; cf. also Gill et al. [2002]

ation problem set, 84.2% are within 1% of the optimal objective value (provided by the other solvers).

3.4.2 Robustness with Non-Probabilistic Uncertainty

Providing flexibility in schedules and temporal plans is viewed as a means to increase their robustness against unforeseen disturbances, such as delays. Several metrics for the flexibility of a schedule have been proposed (see Section 2.3) as well as algorithms for finding high-flexibility schedules [Aloulou and Portmann, 2003; Policella et al., 2009; Banerjee and Haslum, 2011].

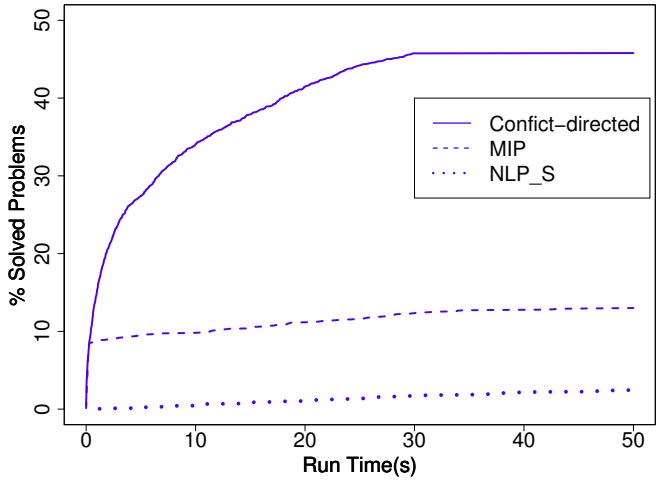
However, flexibility does not necessarily imply robustness: this depends on how "robustness" itself is defined. In abstract terms we may define robustness as the greatest level of disturbance (deviation from expected outcomes) at which the schedule is still successfully executed. (If we assume a probability distribution over deviations is given, the level of disturbance at which the schedule breaks equates to the probability of it breaking during execution. We consider this case in a later section.) To operationalise this definition, we have to specify what kind of disturbances are considered, and how the schedule executive can use flexibility to cope with them. Here, we exemplify by assuming (1) that the possible disturbances are deviations in the time taken to execute an activity from its normal duration, and (2) a partial-order schedule with a dynamic execution strategy.

A partial-order schedule (POS) consists of a set of time constraints between activities such that any realisation that meets these constraints is also resource feasible. In the deterministic case, where the duration of each activity i is a constant d_i, the POS can be represented as an STN with time points t_{s_i} and t_{e_i} for the start and end, respectively, of each activity. Assuming the duration of each activity can vary within some bounds, $[l_{s_i,e_i}, u_{s_i,e_i}]$, the schedule can be modelled as an STPU where the link $e_{s_i e_i}$ from each activity's start to its end is contingent, while remaining time constraints are requirement links. Thus, given a POS we can ask, what is the maximum deviation (i.e., the width of the contingent bound) on any activity in which the STPU is dynamically controllable? This defines our measure of robustness. To compute it, we solve the following problem:

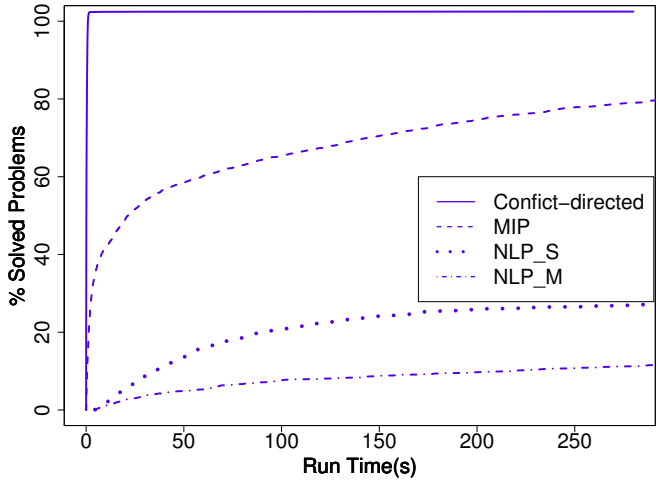
$$
\begin{array}{lll}
\max & \Delta & \\
\text{s.t.} & l_{s_i,e_i} = d_i - \delta_i \geq 0 & \forall i \\
& u_{s_i,e_i} = d_i + \delta_i & \forall i \\
& 0 \leq \Delta \leq \delta_i & \forall i \\
& \text{POS constraints (requirement links)} & \\
& \text{dynamic controllability (3.1)–(3.7)} &
\end{array}
$$

As explained above, requirement link bounds must be allowed to shrink, but their outer limits can be set to the given POS constraints. Since contingent links represent durations, a hard lower bound $L_{s_i e_i} = 0$ applies.

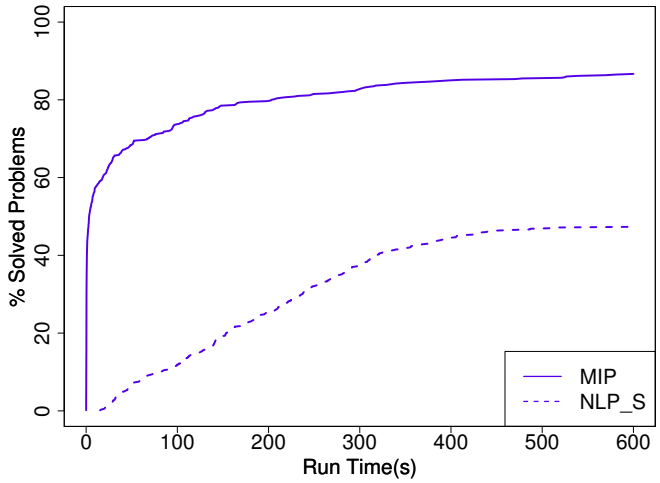
We can also define a one-sided variant of this robustness metric, accounting for delays only, by fixing $l_{s_i,e_i} = d_i$ (i.e., adding deviations only to the upper bound).



(a) minimum relaxation problems



(b) schedule robustness (maximum delay) problems



(c) minimum flexibility problems

Figure 3.8: Runtime distributions for three different solvers (conflict-directed relaxation (CDRU), the MIP model solved with Gurobi and the non-linear model solved with SNOPT) on three problems.

3.4.2.1 Comparison of Solvers

We compared solvers on the one-sided (maximum delay) variant of the problem. As test cases, we use 3400 partial-order schedules for RCPSP/max problems [Kolisch and Padman, 2001] with 10–18 jobs[4]. The schedules are generated by a scheduler that optimises a measure of POS flexibility [Banerjee and Haslum, 2011]. The STPU representation of a schedule has a time point for the start and end of each activity, as described above. Hence, the number of nodes and contingent links is determined by the number of jobs, but the number of (given) requirement links varies from 50 to 300.

The adapted CDRU algorithm is very effective for this problem, and the relative runtimes of the MIP and non-linear solvers, as shown in Figure 3.8(b), are similar to the previous case study. The non-linear solver frequently fails to find optimal solutions. To remedy this issue we run SNOPT repeatedly, using the last solution as the starting point for the next iteration and using its objective value as a lower bound. This configuration (labelled "NLP_M" in Figure 3.8) is more time-consuming, but improves final solution quality: 93% of solutions found by NLP_M are optimal, compared to only around 70% for a single run of the solver.

3.4.2.2 Strong vs. Dynamic Controllability

We can evaluate the robustness of a schedule under strong controllability as well. This requires only solving a linear program. For 84.3% of the test cases, the value of the maximum delay metric is the same. However, 6.38% of cases allow zero delays (i.e., have no robustness) under strong controllability but admit a non-zero value with dynamic execution.

3.4.3 Minimising Flexibility

Although flexibility may improve robustness, there can also be a cost associated with it. For example, to permit a job to start at any point in an interval it may be necessary to reserve resources for the entire time from its earliest starting time to its latest ending time, an interval that may be much larger than the duration of the job. Thus, for a given, fixed level of temporal uncertainty, we may seek the smallest flexibility – meaning the tightest requirement bounds – that is sufficient to maintain dynamic controllability. This is the application that motivated by Wah and Xin [2004]. Their formulation is:

$$
\begin{array}{lll}
\min & \displaystyle\sum_{e_{ij} \in EC} (u_{ij} - l_{ij}) & \\
\text{s.t.} & L_{ij} = l_{ij} \leq u_{ij} = U_{ij} & e_{ij} \in EU \\
& L_{ij} \leq l_{ij} \leq u_{ij} \leq U_{ij} & e_{ij} \in EC \\
& \text{dynamic controllability (3.1)–(3.7)} &
\end{array}
$$

[4]Set J10 from PSPLIB (http://www.om-db.wi.tum.de/psplib/), plus additional problems generated to have more max time lag constraints and allow more variation in resolving resource conflicts.

where L_{ij} and U_{ij} are the bounds of the input STPU. That is, contingent link bounds are fixed, and requirement link bounds may only be tightened.

3.4.3.1 Comparison of Solvers

We use a set of 600 random STPUs of the "GRID" family, generated by the same method used by Wah and Xin [2004]. The number of nodes ranges from 41 to 201, contingent links from 6 to 63, and requirement links from 60 to 360. The runtime distribution is shown in Figure 3.8(c). Since there is no CDRU implementation for this problem, we compare only the MIP and non-linear solvers.

3.4.4 Robustness with Probabilistic Uncertainty

It is a natural extension of the STPU model to associate probabilities with the outcome (timing) of uncontrollable events [Tsamardinos, 2002; Fang et al., 2014]. As mentioned earlier, this allows us to define robustness as the probability of successful plan or schedule execution.

In the probabilistic STN (pSTN) model proposed by Fang et al. [2014] the duration of each contingent link e_{ij} (i.e., the difference $t_j - t_i$) is a random variable D_{ij}. The model makes no assumption about independence or the distribution of these variables.

It is straightforward to transfer the STPU representation of a partial-order schedule, described above, to a pSTN. Since the random variable $D_{s_i e_i}$ represents the duration of an activity we assume it is non-negative. The probability of a successful dynamic schedule execution is then at least

$$\begin{array}{lll} \max & \mathbf{P}\left(\bigwedge_i l_{s_i e_i} \le D_{s_i e_i} \le u_{s_i e_i}\right) & \\ \text{s.t.} & 0 \le l_{s_i, e_i} \le u_{s_i, e_i} & \forall i \\ & \text{POS constraints (requirement links)} & \\ & \text{dynamic controllability (3.1)–(3.7)} & \end{array}$$

The objective value is a conservative (lower) bound on the success probability because it is the probability that all uncontrollable events fall within the chosen bounds. The schedule may still execute successfully even if some durations fall outside these bounds, as the outcomes of other events may fortuitously compensate so that no constraints are violated.

The objective function form depends on the probability distributions over activity durations. For example, if each $D_{s_i e_i}$ is uniform over an interval, we can conservatively approximate the success probability with a linear function. Other distributions give rise to non-linear objectives.

	Maximum Delay			Prob. of Success		
	$>$	$=$	$<$	$>$	$=$	$<$
fluidity	291	332	125	185	441	77
improved flex	476	513	192	334	698	141

Table 3.1: Correlation between schedule flexibility, as measured by *fluidity* [Aloulou and Portmann, 2003] and *improved flex* [Wilson et al., 2014], and robustness. Each entry is the number of instances in which the schedule with a higher fluidity/flex score has a higher ($>$), equal ($=$) or lower ($<$) max delay and the probability of success, respectively, compared to that of the schedule with lower fluidity/flex.

3.4.4.1 Flexibility vs. Robustness

As we are now able to compute different measures of schedule robustness, we can put to test the hypothesis that different measures of schedule flexibility correlate with robustness. We consider two flexibility metrics: *fluidity*, defined by Aloulou and Portmann [2003], and *improved flex*, defined by Wilson et al. [2014]. Both metrics are averages of some form of temporal slack in the schedule.

We use the same set of RCPSP/max problems as before. For each of the two metrics, we take the schedules with the least and greatest fluidity/flex score for each instance. (Of course, we can only use instances for which the scheduler found at least two schedules with different fluidity/flex values.) For both schedules in each such pair, we compute the non-probabilistic (maximum delay) and probabilistic measures of robustness (where this is possible within reasonable time). For probabilistic robustness, we use uniformly distributed durations over an interval between $\pm 30\%$ of the nominal activity duration. We then count how often the schedule that has the higher score according to each flexibility metric has a higher, lower or equal robustness score, according to each of the two robustness measures. The results are summarised in Table 3.1. If there was no correlation between flexibility and robustness, we should find roughly equal proportions of schedules with higher and lower robustness scores. A simple binomial test shows that the observed result is extremely unlikely under this hypothesis. Thus, we conclude that there is a (positive) correlation between fluidity/flex and robustness, although it is quite weak.

3.4.5 Dynamic Controllability with Chance Constraints

Fang et al. [2014] argue that in many situations, minimising risk (probability of failure) is too conservative, and may compromise other objectives (such as cost) too much. Instead, users may prefer to give an absolute bound on risk and optimise other metrics subject to this constraint. They propose a pSTN optimisation algorithm

subject to strong controllability and the chance constraint

$$\sum_{e_{ij} \in EU} (1 - \mathbf{P}(l_{ij} \leq D_{ij} \leq u_{ij})) \quad \leq \quad \rho. \tag{3.17}$$

This makes use of the union bound (Boole's inequality), so it is a conservative estimate of risk: That is, it ensures the probability of violating a requirement is ρ *or less*.

Combining the dynamic controllability model (3.1)–(3.7) with (3.17) enables us to find dynamic execution strategies under chance constraints. Whether the constraint is linear or non-linear, and hence which solvers can be applied, depends on the probability distributions, as noted above.

3.4.5.1 Dynamic vs. Strong Controllability

Since a strongly controllable network is also dynamically controllable, the optimal value of any objective functions can only improve as we consider dynamic instead of strong execution strategies. We can now evaluate how much it improves, by comparing the quality of solutions obtained under dynamic and strong controllability constraints on the test cases used by Fang et al. [2014]. The objective is to minimise makespan under chance constraints. Since the problems feature normally distributed uncertain durations, only the non-linear solver is able to tackle them.

Figure 3.9 shows the distribution of the improvement, measured by the reduction in makespan achieved under dynamic controllability from that of the optimal strongly controllable solution, expressed as a fraction of the latter. (Strongly controllable solutions to chance-constrained problems are generated with Fang, Yu and Williams' solver.)

This clearly shows the value of using a dynamic execution strategy. (In fact, the possible improvement may be even greater since solutions returned by the non-linear solver are often not optimal.) 8.5% of problems are infeasible under strong controllability constraints – that is, no strong (unconditional) execution strategy exists – but are feasible under dynamic controllability constraints. On the other hand, optimisation with dynamic controllability is harder: the NLP solver failed to find a solution for 33% of the instances.

3.5 Conclusions

In many cases where an STPU is used to represent a plan or schedule, the ability to optimise a function of STPU time bounds subject to the constraint that the network is dynamically controllable make it far more useful than being able only to test if dynamic controllability holds. For example, we showed how it enables optimisation of a chance-constrained pSTN under dynamic controllability, leading to better solutions than possible if strong controllability is imposed. It also enabled us to measure schedule robustness in new ways, and thereby test the hypothesis that greater

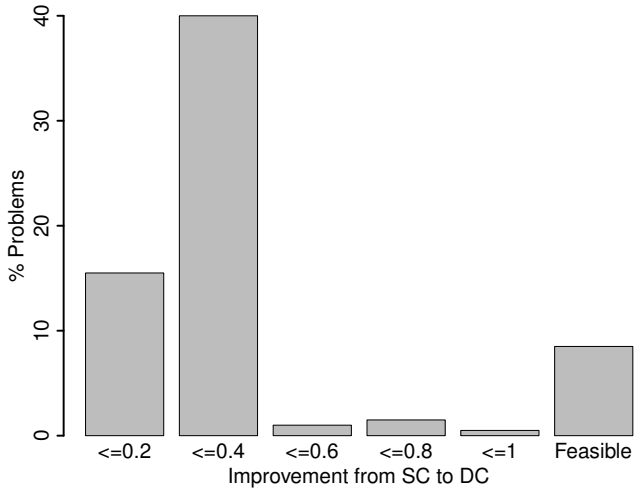


Figure 3.9: Reduction in makespan achieved with dynamic as opposed to strong controllability. Instances in the last column are infeasible under strong controllability but have a valid dynamic execution strategy.

flexibility leads to more robust schedules.

Dynamic controllability is a disjunctive linear constraint. Comparing solution approaches, we found that dealing with disjunctions explicitly, as in the CDRU algorithm [Yu et al., 2014], is the most efficient. However, the MIP and NLP formulations are very flexible, allowing controllability to be combined with other constraints. Our constraint model follows closely the reduction rules of the algorithm by Morris et al. [2001].

Chapter 4

Dynamic Controllability of CCTPU

Yu et al. [2014] extended the STPU to the Controllable Conditional Temporal Problem with Uncertainty (CCTPU) by considering controllable choices as discrete variables. In CCTPU, the links with a label, which is a conjunction of assignments of discrete variables, are activated by those assignments. Yu et al. [2014] introduced a relaxation method to solve over-constrained CCTPU, by making a static assignment of choice variables and relaxing time constraints to produce a dynamically controllable STPU.

In this chapter, to implement the original intent of dynamic control, we introduce the definition of dynamic controllability of a CCTPU that considers making assignments of both time points and discrete choices dynamically by extending it to the discrete variables of the CCTPU. From Yu et al. [2014], we borrow the idea of using dynamic controllability checking algorithms of STPU (See Section 2.2) to find conflicts, which represent the reason why an STPU is not dynamically controllable but extend it to extract a complete set of conflicts.

When implementing the dynamic controllability checking process, some assumptions are needed: (1) each discrete choice is made at one time point, which is *no later than* any other time points related to the choice; (2) only the uncontrollable events *definitely* completed before the time point to make a choice can be treated as an observable condition; and (3) each discrete choice is made following the observation of one, or a sequence of, uncertain events. These assumptions are more conservative than the original concept of dynamic controllability. Our dynamic controllability checking algorithm for a CCTPU is sound but only complete with assumptions. It guarantees to find dynamic strategies under these assumptions.

Last but not least, a similar problem – the dynamic controllability of Conditional Simple Temporal Networks with Uncertainty (CSTNU) – has been studied by Hunsberger et al. [2012]; Combi et al. [2014], which consider conditions as uncontrollable and observable propositions. But we only discuss controllable discrete variables that do not depend on observations.

The work in this chapter extends the conference publication [Cui and Haslum, 2017] by completing the theoretical definitions and validation of the algorithms.

We start by illustrating an example in Section 4.1. In Section 4.2, we introduce the problem statement that includes the definitions CCTPU, dynamic assignment of discrete variables with assumptions and dynamic controllability of CCTPU. We

briefly review conflict resolutions of STPU in Section 4.3.1, which is the basis of our algorithm that extracts a complete set of conflicts (Section 4.3.2) in order to get the DC envelope of an STPU (Section 4.3.3). In Section 4.4, we illustrate how to check dynamic controllability of a CCTPU by aggregating and checking its dynamically controllable envelopes. Section 4.5 shows the correctness and completeness of the algorithms. Section 4.6 and 4.7 are experiments and conclusions, respectively.

4.1 An Illustrative Example

To illustrate the motivation for dynamic controllability of a CCTPU, we take the example of Mr. P's travel plan after work. After leaving work at 5pm, Mr. P is going grocery shopping before having dinner, then catching a bus home. Shopping may take 30 to 50 minutes, depending on how crowded it is. For dinner, Mr. P has two options: He can have a quick dinner at KFC, which only takes 20-30 minutes, or go for his favourite steak. This takes longer, 40-60 minutes, but the restaurant is closer to the bus stop. The bus leaves at 6.50pm. Mr. P neither wants to miss this bus, which will make him wait an hour for the next bus nor arrive at the bus stop before than 6.40pm, to avoid staying out in the cold weather. Mr. P needs to decide his schedule for these two hours.

The CCTPU in Figure 4.1 models Mr. P's problem. The discrete variable c_1 models the choice of dinner ($c_1 = K$ for KFC and $c_1 = S$ for steak). Contingent links (dashed lines: $E1 \to E1'$, $E2 \to E2'$ and $E3 \to E3'$) represent uncertainty, such as the time it takes to shop and have dinner. Those durations are not decided by Mr. P but depend on factors outside his control. The other links express constraints on the solution, in the form of bounds on the difference between two time points. Some links have labels, which are the assignments of the discrete choice variable that active those links. For example, constraints $E1' \to E3$, $E3 \to E3'$ and $E3' \to E$ only need to be met if $c_1 = K$.

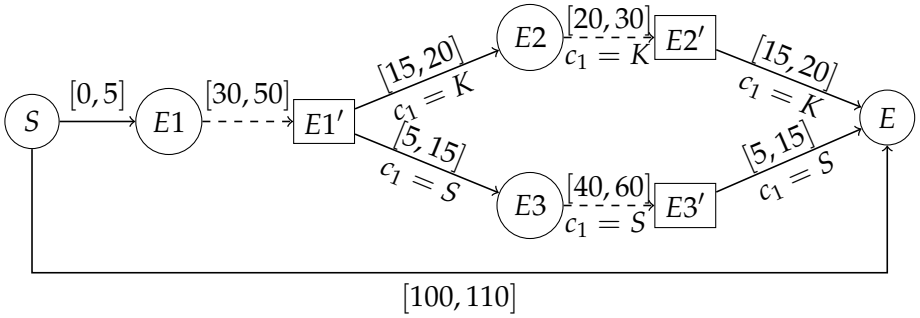


Figure 4.1: The CCTPU of Mr. P's travel problem.

Unfortunately, because of the uncertainty, neither of the two dinner options, if chosen in advance, lead to a schedule that is guaranteed to satisfy Mr. P's requirements. If he goes to KFC and the uncontrollable links E1 and E2 take their lower

bounds, the total duration from S to E is a maximum of 95 minutes, breaking the lower bound of the S–E requirement link. If he goes for steak and links E1 and E3 take their upper bounds, the minimum time from S to E is 2 hours, breaking the upper bound instead. But Mr. P does not have to decide on dinner when he leaves work. He only needs to make it after buying groceries. If he spent more than 40 minutes shopping, he could choose KFC; otherwise, he can enjoy the steak and catch the bus on time.

Although artificial, the example shows the benefit of postponing decisions, including decisions about discrete choices, when faced with uncertainty. (Indeed, that is the motivation for the concept of dynamic controllability initially proposed for the STPU.) There may be no fixed assignment of discrete variables that is feasible, while making those assignments during execution based on observations of uncontrollable events may provide a feasible strategy.

4.2 Problem Definitions

In this section, we introduce theoretical definitions on the dynamic controllability of Controllable Conditional Temporal Problems with Uncertainty (CCTPU). The CCTPU extends the STPU with controllable discrete choices Yu et al. [2014]. We adopt their definition, which is Definition 2.5 in Section 2.1.4, but omit the reward and cost functions since, in this chapter, we consider its controllability only.

The following content of this section defines the solution of a CCTPU and different levels of controllability of the CCTPU.

4.2.1 Preliminary Definitions

Before defining the dynamic controllability of a CCTPU, we introduce the definitions of schedules, execution strategies and strong controllability of a CCTPU.

The following definitions extend concepts from the STPU [Vidal and Fargier, 1999; Morris et al., 2001; Hunsberger, 2009] to the CCTPU.

Definition 4.1. A **schedule** S for a CCTPU is a tuple $\langle A, T \rangle$.

- A is an assignment of each discrete variable c to a value in its domain, i.e., $A(c) \in D(c), \forall c \in C$. A link $e \in E$ is **activated** if $A \models \ell_E(e)$; A link $e \in E$ is **deactivated** if $A \not\models \ell_E(e)$.
- T is a mapping $T : V \to \Re^+ \cup \{0\}$, where $T(v)$ is the scheduled time of time point $v \in V$.

A schedule S is **consistent** if it satisfies all the constraints of links activated by its assignments of discrete variables.

Definition 4.2. A **projection** p of a CCTPU is constructed by replacing every uncontrollable link $eu_i = [l_i, u_i]$ in EU by the singleton $eu_i = [p_i, p_i]$, where $p_i \in [l_i, u_i]$.

Each projection of a CCTPU is a possible outcome of uncertainties that may occur, and it is a CCTP.

Definition 4.3. An **execution strategy** for a CCTPU is a tuple $\langle DT, ES \rangle$, where

- $DT : C \to V \cup \{0\}$ maps each discrete variable c to the time point $DT(c)$ at which the choice for c will be made, the domain of $DT(c)$ is the union of all nodes and the beginning time of the network and
- $ES : P \to S$ is a mapping from the set P of all projections of the CCTPU to the set S of schedules.

An STPU is a special CCTPU without discrete variables, and its *execution strategy* ES is **viable** iff $ES(p)$ is consistent for every projection $p \in P$. Based on the above, Vidal and Fargier [1999] introduced three levels of controllability for the STPU: weak, strong and dynamic. Extending strong controllability to the CCTPU is straightforward:

Definition 4.4. A CCTPU is **strongly controllable** when there is execution strategy $\langle DT, ES \rangle$ such that $DT(c) = 0$ for all $c \in C$, ES is viable, and satisfying $ES(p_1)(x) = ES(p_2)(x)$ and $ES(p_1)(c) = ES(p_2)(c)$ for each controllable time point x, discrete variable c and any two projections p_1 and p_2.

Strong controllability means there is a universal schedule which satisfies all constraints in every projection of the problem. This means the schedule can be made before execution.

Yu et al. [2014] define a dynamically controllable solution of a relaxed CCTPU as a fixed assignment of the discrete variables such that the resulting STPU is dynamically controllable. Specifically, there is a viable execution strategy $\langle DT, ES \rangle$ such that

- $DT(c) = 0$ for all $c \in C$, and
- for any two projections p_1 and p_2,
  - $ES(p_1)\{\prec t\} = ES(p_2)\{\prec t\} \Rightarrow ES(p_1)(x) = ES(p_2)(x)$ for each controllable time point x, $t = ES(p_1)(x)$ and
  - $ES(p_1)(c) = ES(p_2)(c)$ for each discrete variable $c \in C$.

That is, the decisions on discrete variables are strongly controllable.

To fully extend dynamic controllability to CCTPU, in the following content of this section, we define the dynamic assignment for discrete variables and dynamic controllability of CCTPU.

4.2.2 Dynamic Assignments for Discrete Variables

In this subsection, we define the dynamic assignment and prehistory for discrete variables of CCTPU. The basic idea of dynamic assignments is to assign values to discrete variables based on past observations as dynamic decisions on timepoints.

As the first thing, we can define the dynamic assignment based on previous definitions.

Definition 4.5. A **dynamic assignment** $A(c) = dc_i$ is made at $DT(c)$ and it deactivates links with labels $c = dc_j, \forall dc_j \neq dc_i$.

In order to make the definition valid in a dynamically controllable strategy, we have to define the prehistory of the assignment and restrict $DT(c)$ with the links having labels related to c.

However, several obstacles prevent us from giving a direct definition of the prehistory: (1) the assignment is not explicitly associated to timepoints; and (2) the contingent links included in the past observations may be completely different for different choices have been made. Due to these difficulties, we have to introduce basic definitions, such as *partial assignments, precedences and conditional precedences* with some assumptions.

Making dynamic choices for controllable discrete variables in a temporal problem is not a new topic. Conrad and Williams [2011] define the dynamic execution for an STN with controllable discrete variables. This dynamic execution decides to activate an event or not in real-time and maintains a consistent labelled value set. We adopt their definition of environment for the STN with discrete variables as a partial assignment of discrete variables.

Definition 4.6. (Partial Assignment) The environment is a **partial assignment** PA of the discrete variables. Before execution, the environment is empty $PA = \emptyset$; after a feasible execution, the environment consists of assignments of all discrete variables $PA = A$.

In execution, the dynamic decisions on discrete variables in a strategy are made at specific timepoints that obey the chronological orders implied by constraints. The observations of contingent links on which these decisions can depend on have to complete before the decision timepoints. As a necessary condition, we introduce the definition of precedence between timepoints in order to represent the set of contingent links that each decision of discrete variable can observe. Furthermore, the precedence definition can also help to describe and find the chronological order to assign discrete variables.

Definition 4.7. (Precede) For any pair of time points $v_i, v_j \in V$, v_i *precedes* v_j, denoted as $v_i \preceq v_j$, iff $S(v_i) \leq S(v_j)$, for every consistent schedule S. For any link $e_i \in E$ and $v_j \in V$, e_i *precedes* v_j, $e_i \preceq v_j$, iff $start(e_i) \preceq v_j$ and $end(e_i) \preceq v_j$; v_j *precedes* e_i, $v_j \preceq e_i$, iff $v_j \preceq start(e_i)$ and $v_j \preceq end(e_i)$.

This definition excludes some labelled precedences. For instance, in the example in Figure 4.2, precedences $A \preceq B$ and $C \preceq D$ hold in every consistent schedule, but precedence $B \preceq C$ and $D \preceq A$ do not exist because they only hold in the environments of $\{c_3 = 1\}$ and $\{c_3 = 2\}$, respectively. With the given precedences, the order of contingent links AB and CD is not decidable, so neither of them is guaranteed to be an observation of deciding discrete variables c_1, c_2 or c_3. Thus, the dynamically controllable decisions of timepoints and discrete variables cannot be assumed to happen after the observation of any of the contingent links, and therefore have to work for any value in their range.

Additionally, we define the conditional precedence under a certain partial assignment PA.

Definition 4.8. In the environment of partial assignment PA, any pair of time points $v_i, v_j \in V$, v_i *precedes* v_j under PA, denoted as $v_i \preceq_{PA} v_j$, iff $S(v_i) \leq S(v_j)$, for every consistent schedule S such that its assignment $A \models PA$. For any link $e_i \in E$ and $v_j \in V$, e_i *precedes* v_j under PA, $e_i \preceq_{PA} v_j$, iff $start(e_i) \preceq_{PA} v_j$ and $end(e_i) \preceq_{PA} v_j$; v_j *precedes* e_i under PA, $v_j \preceq_{PA} e_i$, iff $v_j \preceq_{PA} start(e_i)$ and $v_j \preceq_{PA} end(e_i)$.

With definition 4.8, the example in Figure 4.2 has conditional precedences $B \preceq_{c_3=1} C$ and $D \preceq_{c_3=2} A$ that enable different dynamic assignments after choosing c_3. Link AB can be the observation to decide c_2, when $c_3 = 1$; Link CD can be the observation to decide c_1, otherwise.

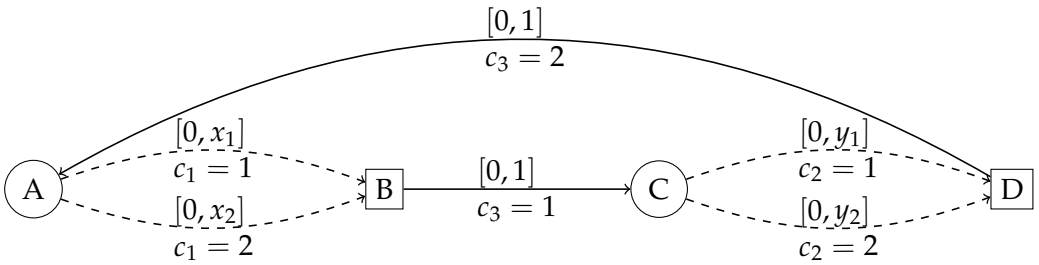


Figure 4.2: An Example Showing Precedences

We introduce an assumption of the decision timepoint of a discrete variable to associate the discrete variable to timepoints and restrict the decision time according to the links with labels including assignments of the variable.

Assumption 4.1. The assignment $A(c)$ is made once, at the time point $DT(c)$, which must occur no later than any link $e \in E$ such that $\ell_E(e)$ mentions c. That is, $DT(c) \preceq e, \forall e$ s.t. $\ell(e) \cap D(c) \neq \emptyset$.

With assumption 4.1, Definition 4.5 means during execution, before $DT(c)$, the strategy is the same for different options of c, after $DT(c)$, the strategy only respects to the sub-network without links deactivated by $A(c)$.

One part of the assumption is implied by the definition of execution strategy, which is the decision time point $DT(c)$ is associated with a single timepoint in V. The other part is the associated decision timepoint should choose among nodes that precede every link activated by c. However, separate decisions that $c \neq dc_i$ can be made at different time points in real execution. For example, if we have the choice of performing a task today, tomorrow, or the day after, we could decide now to not do it today without committing to which of the other two days it will be done. To some extent, this limitation can be recovered by remodelling the problem. For example in Figure 4.3, the variable with three options can be remodelled to the CCTPU in Figure 4.4 which adds a dummy node of A as A' and contains two variables with two options each.

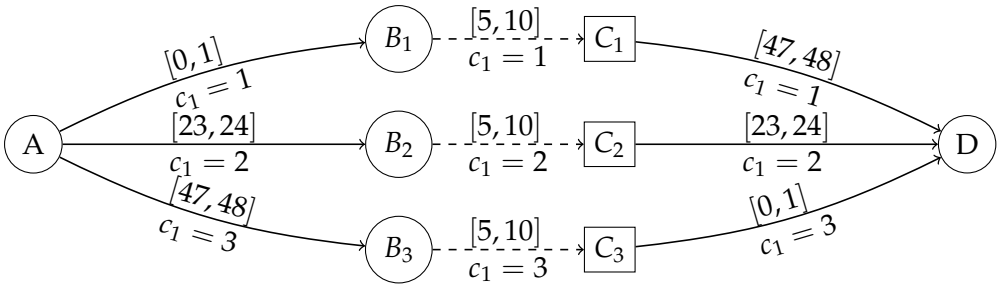


Figure 4.3: Example of a CCTPU with a variable with three options

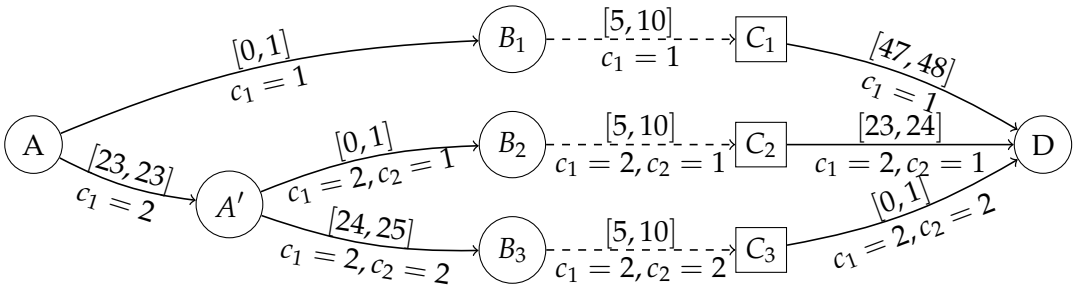


Figure 4.4: Example of Remodelling

For dynamic controllability of an STPU, the *observed situation* [Vidal and Fargier, 1999], or *prehistory* [Morris et al., 2001; Hunsberger, 2009], at any time consists of the observed durations of contingent links that have finished before that time. Given a schedule S of STPU, the prehistory of a time point x is

$$S\{\prec x\} = \{p_{ij} | S(v_i) + p_{ij} \leq S(x)\},$$

where p_{ij} is the observed duration of contingent link e_{ij}. However, we restrict the prehistory of discrete variables to only those contingent links that must finish before the variable's decision timepoint, so that the set of contingent links in the prehistory is stable with a given partial assignment.

Assumption 4.2. Given a projection p under a certain partial assignment PA, the **prehistory** of a discrete variable c is the observed durations of contingent links which **must** be activated by PA and finish before or at $DT(c)$ in every execution, denoted $P_{PA}(p)\{\preceq c\} = \{p_{ij} | e_{ij} \preceq_{PA} DT(c)$ and $PA \models \ell(e_{ij}), p_{ij} \in p\}$.

Furthermore, under a certain partial assignment, the **prehistory** of a set of unassigned discrete variables is the union of the prehistories of the variables in the set,

$$P_{PA}\{\prec C_s\} = \bigcup_{c \in C_s} P_{PA}\{\prec c\}.$$

Different from the observation of timepoint scheduling, contingent links that may end, but must not yet end, before the decision time point of a discrete variable is not

observations making the decision. In the DC reduction for an STPU, in a triangle ABC with contingent link AC, if the contingent link AC that may end but must not end before a timepoint B, it indicates that B is in the unorder case in triangle reduction. Thus, B has a wait constraint with AC. It means that B can be used as an observation in the execution of STPU. The extended work to relax this assumption which considers those contingent links as observations is one of our future work.

Therefore, based on the prehistory which is the projections of contingent links that must finish before the decision timepoint, the dynamic assignment of a discrete variable is the choice made at its decision timepoint which must precede every link with labels related to its assignment, and the decision activates and deactivates subnetworks accordingly.

4.2.3 Dynamic Controllability of CCTPU

The dynamic controllability means there is a viable execution strategy in which each decision only depends on observations before the decision. Based on the definition of dynamic assignment of discrete variables with assumptions, we provide a definition of dynamic controllability of CCTPU. The definition is sound but not complete regarding the intent of dynamic controllability, but it is complete with the assumptions. Additionally, ways to relax assumptions are discussed at the end of this subsection.

Definition 4.9. A CCTPU execution strategy $\langle DT, ES \rangle$ is **viable** iff

- $DT(c)$ precedes the start of every link $e \in E$ such that c appears in $\ell_E(e)$
- and $ES(p)$ is consistent for every projection p.

We now define dynamic controllability of a CCTPU as follows.

Definition 4.10. A CCTPU is **dynamically controllable** if there is a viable execution strategy $\langle DT, ES \rangle$ such that for any two projections p_1 and p_2,

- $ES(p_1)\{\prec t\} = ES(p_2)\{\prec t\} \Rightarrow ES(p_1)(x) = ES(p_2)(x)$, where $t = ES(p_1)(x)$, for each controllable time point x,
- $P_{PA}(p_1)\{\prec c\} = P_{PA}(p_2)\{\prec c\} \Rightarrow ES(p_1)(c) = ES(p_2)(c)$, for each discrete variable c and partial assignment PA.

In this definition, the decisions on timepoints are the same as dynamic controllability of STPU, the decisions on discrete variables only depend on the prehistories before the scheduling time of their decision timepoints, which also obey the origin of dynamic controllability.

4.2.4 Dynamically Controllable Envelopes

Definition 4.10 only helps to check dynamic controllability of a given CCTPU. We present the following definitions of **dynamically controllable envelopes** to explain our dynamic controllability checking algorithm in this chapter.

Dynamically controllable envelopes intend to define the preconditions under which, the following CCTPU has a dynamically controllable execution strategy. In other words, the subset of uncertain prehistory with which the following CCTPU is dynamically controllable is the dynamically controllable envelope, which can be regarded as a relaxation of contingent links in the prehistory. For example, in the illustrative example shown as Figure 4.1, the subset $[30, 40]$ of event $E_1 \to E_1'$ is the dynamically controllable envelope of making decision $c_1 = S$ and $[40, 50]$ is the envelope of making decision $c_1 = K$.

The dynamically controllable envelope of a CCTPU can be aggregated from fully assigned branches which are STPU. In the illustrative example, if we combine the two envelopes $[30, 40]$ and $[40, 50]$, it covers the whole uncertain prehistory $E_1 \xrightarrow{[30,50]} E_1'$. Thus, a dynamic choice based on the observation of $E_1 \to E_1'$ can make the whole strategy feasible and dynamic.

The dynamically controllable envelope of a fully assigned CCTPU is the set of relaxations that can make the STPU dynamically controllable. A relaxation reduces the uncertainty by increasing lower bounds and/or decreasing upper bounds of contingent links. In the dynamically controllable envelope of a fully assigned CCTPU, the relaxable links are all contingent links not deactivated by its assignment. One relaxation EU' of a given set of contingent links EU can be formulated as for all $eu \in EU$, there exists $eu' \in EU'$, such that $eu' \subseteq eu$ and eu' and eu start from and end at a same pair of nodes.

$$EU' = relaxed(EU) = \{eu'_{ij} = [l'_{ij}, u'_{ij}] | \forall eu \in EU, eu = [l_{ij}, u_{ij}], l_{ij} \leq l'_{ij} \leq u'_{ij} \leq u_{ij}\}. \tag{4.1}$$

Given a relaxation EU' and the CCTPU N, an updated network

$$N' = updated(N, EU') = N \setminus EU \cup EU'$$

which replaces the set of contingent links by the relaxation.

Definition 4.11. The **dynamically controllable envelope** of a CCTPU with a full assignment A is $Env(A) = \{EU' | EU' = relaxed(EU), N' = updated(N, EU'), N'$ is a dynamically controllable STPU under assignment $A.\}$, where EU' is a relaxation of EU.

Envelopes with a common prehistory can be aggregated. For instance, two envelopes with full assignments $Env(A_1)$ and $Env(A_2)$, where $A_1 = PA \cup \{c = d_1\}$ and $A_2 = PA \cup \{c = d_2\}$ have the common partial assignment PA, aggregate to $Env(PA, c)$ on prehistory $P_{PA}\{\prec c\}$. The combined envelope $Env(PA, c)$ describes the previous condition, in which one of the options of c can make a dynamically controllable strategy.

Definition 4.12. The **dynamically controllable envelope** of assigning a discrete variable c in a CCTPU after a certain partial assignment PA is $Env(PA, c) = \{EU' | EU' = relaxed(EU, PA, c), N' = updated(N, EU')$ is dynamically controllable.$\}$, where $EU' =$

$relaxed(EU, PA, c)$ is the set of relaxed bounds of contingent links belong to $P_{PA}\{\prec (PA \cup c)\}$ and make the updated CCTPU dynamically controllable.

After aggregating envelopes from all branches $Env(PA \cup \{c = d_i\})$ to $Env(PA, c)$, we want to aggregate $Env(PA \setminus \{c' = d'\})$, where $\{c' = d'\} \subset PA$. The combined envelope $Env(PA, c)$ cannot be used directly in the next level aggregation that combining branches for discrete variable $c' \in PA$ because it may contain links belong to $P_{PA}\{\prec c\}$ not $P_{PA\setminus c'}\{\prec c'\}$. Thus, we present Definition 4.13.

Definition 4.13. The **dynamically controllable envelope** of a CCTPU under a certain partial assignment PA is $Env(PA) = \{EU'|EU' = relaxed(EU, PA) \wedge N' = updated(N, EU')$ is dynamically controllable$\}$, where $relaxed(EU, PA)$ is the set of relaxed bounds of contingent links where relaxable links belong to $P_{PA}\{\prec PA\}$ and make the updated CCTPU dynamically controllable.

Therefore, the aggregating process to achieve the envelope of CCTPU with partial assignment PA before assigning the next discrete variable c is the set of unions of selected subsets of envelopes of branches $PA'_d = PA \cup \{c = d\}$, which is

$$Env(PA, c) = \{ \bigcup_{d \in D(c)} EU'_d | EU'_d \in Env(PA'_d)\}. \tag{4.2}$$

Next, envelope $Env(PA)$ selects a subset of $Env(PA, c)$ in the selection rule $eu'_x = eu_x | \forall eu_x \notin P_{PA}\{\prec PA\}$ that selects relaxations whose relaxed links are in the prehistory of PA.

$$Env(PA) = select_{eu'_x = eu_x | \forall eu_x \notin P_{PA}\{\prec PA\}} Env(PA, c) \tag{4.3}$$

The envelope of a CCTPU can be collected by implementing the aggregation in Equation 4.2 repeatedly until $PA = \emptyset$,

A CCTPU is dynamically controllable if and only if its dynamically controllable envelope with an empty partial assignment is the set of the original bounds,

$$Env(\emptyset) = EU.$$

Additionally, the dynamically controllable envelopes of CCTPU satisfy the following equation,

$$Env(A) = EU \Rightarrow Env(PA) = EU \Rightarrow Env(\emptyset) = EU. \tag{4.4}$$

This equation means a CCTPU is dynamically controllable if it has a dynamically controllable envelope with a partial assignment, which covers the uncertainty in the prehistory of the partial assignment, and if a CCTPU has a dynamically controllable STPU branch, it is dynamically controllable.

Last but not least, we introduce the last assumption.

Assumption 4.3. For each discrete variable c, the end point of a contingent link or timepoint 0 can be $DT(c)$.

The rest of the nodes cannot be decision timepoints based on current definitions because different decisions are made according to different observations. However,

the observations of making assignments at those nodes are the same as at some other nodes. For instance in Figure 4.5, dashed lines are contingent links, and the diamond node is the latest decision timepoint. If the decision timepoint is at r_2, the observation set only contains the prehistory before t_1. Thus, potential decision timepoints are S, t_1, t_2 and the diamond node.

This assumption guarantees that the observation differs when making choice c. We call this different observation the key observation to assign c. Before deciding on c, the dynamically controllable execution strategy makes consistent decisions regardless of the following options, which means decisions before the key observation are the same for different options of c. However, after observing the key contingent link, different following decisions are made depending on the options of c.

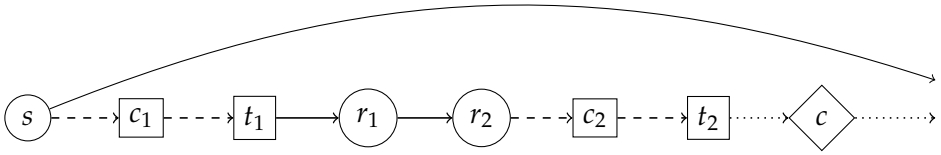


Figure 4.5: Alternatives for $DT(c)$. Squares are uncontrollable time points, circles controllable time points and the diamond is the latest decision time of c.

However, Assumption 4.3 does rule out strategies that wait for parallel contingent links to finish before making a decision.

4.3 Extracting Dynamically Controllable Envelopes of STPU

Given an STPU, the approach in this section returns its dynamically controllable envelope. The dynamically controllable envelope of an STPU is the observable condition under which the STPU is dynamically controllable. Finding the dynamically controllable envelope of an STPU is to find the subset of situations in which no conflict of dynamic controllability occurs.

Both Morris's $O(n^4)$ and $O(n^3)$ algorithms terminate after finding a conflict which is a semi-reducible negative cycle. Yu et al. [2014] solve over-constrained STPUs with a conflict-directed search method. It learns conflicts for dynamic controllability by recording support links, which are the original edges, in propagations and consistency checks of Morris's $O(n^4)$ dynamic controllability checking algorithm. It can learn one conflict each time. By iteratively finding and resolving conflicts, a relaxed and dynamically controllable STPU instance can be found. However, the solution of the iteratively determining method is a dynamically controllable instance under a particular objective function (e.g. minimising relaxation cost), its conflict resolutions may lead to an un-dynamically controllable instance under other objective functions. Therefore, we need to get all necessary conflict resolutions whose solution space equals to the set of dynamically controllable solutions to get the dynamically controllable envelopes.

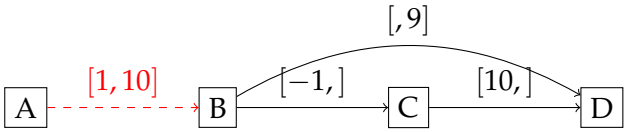


Figure 4.6: An STPU contains a conflict. Because the temporal constraints on BD and CD infer that $u_{BC} \leq -1$, which means C has to be scheduled before the observation of B. Thus, triangle ACB is in the precede case, and the upper bound of AC $u_{AC} \leq l_{AB} - u_{CB} = 0$. However, if C is scheduled no later than A and the uncertain duration of AB is greater than 1, the requirement link on BC cannot be satisfied.

In this section, we start with illustrating the current conflicts learning algorithm. Then we introduce a new method modified from Morris's $O(n^3)$ dynamic controllable algorithm that can extract all conflicts and provide the formulation of the dynamically controllable envelopes for the STPU.

4.3.1 Conflict Resolutions of STPU

Finding the dynamically controllable envelope of an STPU is to find the subset of prehistories in which all conflicts do not exist. This problem is similar to relaxing an over-constrained (non-DC) problem. Yu et al. [2014] formulate the relaxation problem as a linear programming model with a set of constraints derived from *conflicts*. The conflicts represent the reasons why the STPU is not dynamically controllable. A conflict is a semi-reducible negative cycle in the network after applying dynamic controllability reduction rules (see Section 2.2.2.2).

A conflict can be represented as follows:

$$\sum_{i \in conf_j} x_i < 0, \tag{4.5}$$

where x_i are the original bounds (l_i or u_i) of links e_i in the conflict. A conflict resolution is a linear constraint

$$\sum_{i' \in conf_j \cap ER} x'_{i'} + \sum_{i \in conf_j \setminus ER} x_i \geq 0 \tag{4.6}$$

where ER is the set of relaxable links and $x'_{i'}$ are variables for the relaxed bounds.

Another way to resolve conflicts with connected lower- and upper-case labels of the same node is to break the reduction, since the cross-case reduction rule (Equation 2.1) has a label condition:

(CROSS-CASE REDUCTION) If $x \leq 0, B \neq C$,
$A \xleftarrow{B:x} C \xleftarrow{c:y} D$ adds $A \xleftarrow{B:(x+y)} D$.

For example, the STPU in Figure 4.6 has a conflict that can be found by any dy-

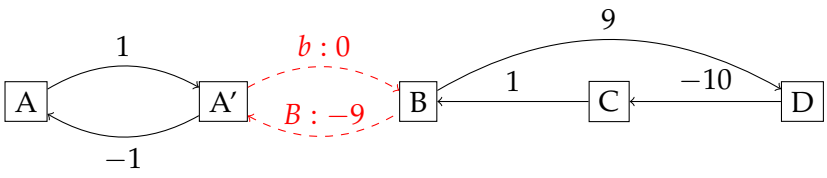


Figure 4.7: The labelled distance graph of Figure 4.6.

namic controllability checking algorithms. The reduction process of Morris's $O(n^3)$ algorithm represented by the labelled distance graph is shown in Figure 4.7. Equation 4.7 illustrates the reduction process. The pair of parentheses shows an added link $C \xleftarrow{0} A$ reduced from the labelled distance graph.

$$A \xleftarrow{-1} A' \xleftarrow{B:-9} B \xleftarrow{1} (C \xleftarrow{-10} D \xleftarrow{9} B \xleftarrow{b:0} A' \xleftarrow{1} A) \tag{4.7}$$

A conflict resolution of the found conflict is

$$-u'_{AB} - l'_{BC} - l'_{CD} + u'_{BD} + l'_{AB} \geq 0.$$

In order to resolve the conflict, the total amount of relaxation is 9.

However, this conflict can also be resolved by only relaxing $C \leftarrow D$ by 1. It will make the back propagation of $C \xleftarrow{-9} D$ stop at B and add a new link $C \xleftarrow{0} B$. Then the back propagation of $A \xleftarrow{-10} B$ is

$$A \xleftarrow{-1} A'(\xleftarrow{B:-9} B \xleftarrow{1} C \xleftarrow{0} B \xleftarrow{b:0} A' \xleftarrow{1} A)$$

which has to stop before $B \xleftarrow{b:1} A$ because the propagated path $A \xleftarrow{B:-9+1} B$ has a label of B which does not satisfy the condition of cross-case reduction. The complete conflict resolution of the conflict in Equation (4.7) is

$$-u'_{AB} - l'_{BC} - l'_{CD} + u'_{BD} + l'_{AB} \geq 0 \vee l'_{CD} + u'_{BD} \geq 0.$$

Therefore, the resolution of a single conflict is a disjunction with one linear constraint of the form 4.6 and other linear constraints over the links whose relaxations can break the reductions:

$$\bigvee_{k \in res_j} \sum_{i' \in res_{jk} \cap ER} x'_{i'} + \sum_{i \in res_{jk} \setminus ER} x_i \geq 0 \tag{4.8}$$

where res_j is the set of conflict resolutions of $conf_j$.

Yu et al. solved an LP over constraints of form (4.6) to find a single relaxation. However, provided we find *all* conflicts in the STPU, the solution of conflict resolution constraints of Equation 4.8 represents the space of all relaxations, which is the same as the dynamically controllable envelope. Extracting all conflicts can be done by

adapting current DC checking methods [Morris, 2006; Shah et al., 2007; Hunsberger, 2013; Nilsson et al., 2013; Morris, 2014] and keeping a record of the conflicts.

4.3.2 Extracting Conflicts of STPU

In this subsection, we describe the method to extract a complete conflict set of a given STPU. Since the conflicts are negative cycles, the number of conflicts is infinite when there is one negative cycle, and the edges can reduce along the negative cycle infinitely. However, our algorithm only finds conflicts that cannot be divided into other found conflicts, which makes the found conflicts a complete conflict set.

Our algorithm is a variation of the dynamic controllable checking algorithm in Morris [2014]. Morris's cubic algorithm identifies a non-dynamically controllable STPU by finding a semi-reducible negative cycle, we will call it negative cycle in this paper, during propagation according to dynamic controllability reduction rules. It propagates every negative link backwards along positive links only, adds a positive link when the propagated path is positive and raises another propagation when meeting a negative link. A conflict is found when the raised propagation is a negative link that has an unfinished ancestor propagation, and the algorithm terminates. It guarantees that every negative link is propagated once at most.

Our approach expands Morris's $O(n^3)$ algorithm by propagating through non-shortest paths and recording a complete set of conflicts. The conflict resulting in the termination can be learned by memorising the path when tracing back.

However, iteratively extracting and resolving one conflict learned from the dynamic controllability checking algorithm cannot get such a complete conflict set, since resolving one conflict may prevent us from finding other conflicts. For example in Figure 4.8, where solid lines are edges and dotted lines are paths consisting of positive links to be propagated. Both paths p_1 and p_2 formulate negative cycles $A \leftarrow B \leftarrow A$, but only the shorter one will be found when calling backpropagation from $A \xleftarrow{L_1:-x} B$. Furthermore, if the conflict resolution relaxes $A \leftarrow B$, it may resolve the other conflict in the instance, but the conflict resolution of the other conflict can never be extracted.

Another example in Figure 4.9 shows a case with cyclic calls of backpropagations. If it first calls backpropagation from $A \xleftarrow{L_1:-x} B$, the algorithm will terminate when finding negative cycle I. Furthermore, if the conflict resolution relaxes the overlapped part of the two negative cycles, it may prevent from finding negative cycle II.

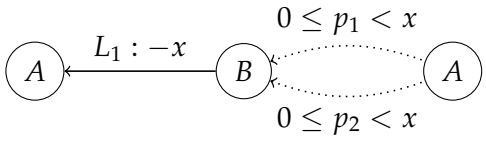


Figure 4.8: Alternative conflicts I

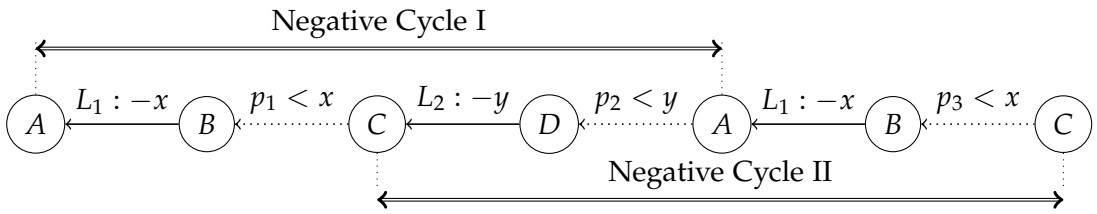


Figure 4.9: Alternative Conflicts II

Based on these observations, to find all necessary conflict resolutions, we have to modify Morris's algorithm in the following ways: (1) replacing the shortest path process by a method that can propagate through all possible paths and (2) propagating some negative links more than once. If the backpropagation of a negative link is involved in a conflict, but the conflict is learned at cyclic propagations ending at another negative link, this propagation may be *incomplete*. For instance in Figure 4.9, if negative cycle I is found by the back propagation starting from $A \xleftarrow{L_1:-x} B$, the raised propagation of $C \xleftarrow{L_2:-y} D$ is not complete which needs to be propagated again. If we start a new round of propagation of $C \xleftarrow{L_2:-y} D$, negative cycle II will be found and the propagation is complete. The propagations without ancestor propagations and involving in no conflict are complete.

The approach to extract a complete set of conflicts is summarised in Algorithms 7, 8 and 9. Algorithm 9 and Lines 7 – 11 of Algorithm 8 are the DFS process, while the rest is the same as Morris's method.

Algorithm 7 calls *BackPropagation* once on every negative edge of the labelled distance graph of a given STPU. *NegPathEnds* records negative paths when tracing back to formulate *NegCycles*.

BackPropagation (Algorithm 8) terminates if the current node recursively reached itself through a negative path (lines 1 – 3) or if propagation from the current node has already been successfully completed (lines 4 – 6). $disEdge[i]$ is the reduced distance edge from i to the end of *srcLink* in the current round of propagation.

In DFS (Algorithm 9), we enumerate edges ending at the current node *crtNode* (Line 2 – 29). Lines 3 – 5 prevent cross-case reductions that do not satisfy the label condition. As proven by Morris's algorithm, the algorithm only propagates through non-negative links ending in *crtNode* (Line 6 – 15). The function $Reduction(X, Y)$ (Line 7) adds two links X and Y based on the dynamic controllability reduction rules (Equation 2.1) and keeps a record of the reduction history. If the propagated path is positive, a new edge is added (Line 9), else, propagation continues to call DFS (Line 13). When encountering a negative link, the algorithm will call *BackPropagation* again (Line 17). *bReturn* marks the existence of a recursively called negative link. If there exists a recursive call, the current path from *crtNode* to *srcNode* can be part of the found conflict, so we add the path $disEdge[crtNode]$ to $NegPathEnds[srcNode]$ in line 19. Furthermore, $NegPathEnds[crtNode]$ keeps a record of negative paths

(as $NodeX \xrightarrow{e1} crtNode$) ending at the current node, which can propagate through $crtNode \xrightarrow{disEdge[crtNode]} srcNode$ and formulate a new negative path as $NodeX \xrightarrow{NewPath} srcNode$ ending at the source node, where $NewPath$ propagates $e1$ through path $disEdge[crtNode]$. Line 23 adds the negative cycle if it can be found in the propagation of $NewPath$; Line 25 adds the negative path, otherwise.

Algorithm 7: Extracting the DC envelope of an STPU.

Algorithm: DCEnvelopeSTPU(N)
Input: An STPU N.
Output: A set of conflicts $NegCycles$

```
1  global LabelN = labelledDistanceGraph(N)
2  NegCycles = ∅
3  for e in negative edges of LabelN do
4  |  NegPathEnds = {∅}
5  |  BackPropagation(e, NegPathEnds, NegCycles)
6  end
7  return NegCycles
```

Algorithm 8: Modified backpropagation process that can extract all conflicts.

Algorithm: BackPropagation($srcLink$, $NegPathEnds$, $NegCycles$)
Input: A negative link $srcLink$, the vector of sets of negative paths end at same nodes $NegPathEnds$, the set of negative cycles $NegCycles$.
Output: Return the propagation result and an updated $NegCycles$

```
1  if ancestor call with same srcLink then
2  |  return False
3  endif
4  if prior successfully terminated call with srcLink then
5  |  return True
6  endif
7  disEdge = {}
8  disEdge[srcLink.start] = srcLink
9  if DFS(srcLink.start, srcLink.end, NegPathEnds, NegCycles) then
10 |  return True
11 endif
12 return False
```

The complexity of our conflict extraction method is $O(E^3)$ in the worst case, when all links are negative and the network is fully connected. It is slower than the state-of-art DC checking methods and CDRU. However, it returns the complete set of conflict resolution constraints, not only whether conflicts exist, or one relaxation. Additionally, it can be improved by cutting unnecessary exploration in the DFS process.

4.3.3 Dynamically Controllable Envelopes of STPU

After finding the complete set of conflicts, the dynamically controllable envelope of an STPU is the solution space of the conflict resolutions. The constraint model is a

Algorithm 9: The DFS process that do reductions through positive links.

Input: Current node *crtNode*, the source node of back-propagation *srcNode*, the vector of sets of negative paths end at same nodes *NegPathEnds* and the set of conflicts *NegCycles*.
Output: Return DFS result and the updated *NegCycles* and *NegPathEnds*
Initialization:

```
1  bReturn = True
```

Algorithm: DFS(*crtNode*, *srcNode*, *NegPathEnds*, *NegCycles*)

```
2  for e ends with crtNode do
3      if e is unusable then
4          continue
5      endif
6      if e.weight ≥ 0 then
7          NewE = Reduction(disEdge[crtNode], e);          // Equation (2.1)
8          if NewE.weight ≥ 0 then
9              LabelN.addEdge(NewE)
10         else
11             TmpDis = disEdge[e.start]
12             disEdge[e.start] = NewE
13             bReturn& = DFS(e.start, srcNode, NegPathEnds, NegCycles)
14             disEdge[e.start] = TmpDis
15         endif
16     else
17         if !BackPropagation(e, NegPathEnds, NegCycles) then
18             bReturn = False
19             NegPathEnds[srcNode].add(disEdge[crtNode])
20             for e1 in NegPathEnds[crtNode] do
21                 NewPath = Reduction(disEdge[crtNode], e1)
22                 if NewPath contains conflicts then
23                     NegCycle.add(NewPath)
24                 else
25                     NegPathEnds[srcNode].add(NewPath)
26                 endif
27             end
28         endif
29     endif
30 end
31 return bReturn
```

conjunction of conflict resolutions.

Given the set of relaxable edges ER that represents the uncertain situations which can be partially considered in the STPU, the dynamically controllable envelope of the STPU is $x'_{i'}$ satisfying the following constraints

$$\bigwedge_{j \in Conf} \bigvee_{k \in res_j} \sum_{i' \in res_{jk} \cap ER} x'_{i'} + \sum_{i \in res_{jk} \setminus ER} x_i \geq 0. \tag{4.9}$$

The dynamically controllable envelope of an STPU can be used to aggregate the

dynamically controllable envelopes of the CCTPU.

4.4 Dynamic Controllability Checking of CCTPU

Central to our algorithm for finding a dynamic execution strategy is the notion of the "envelope" of a partial assignment of the discrete variables.

Definitions of envelopes in Section 4.2.4 can also be rephrased as the following definition.

Definition 4.14. Given a partial assignment to a subset of discrete variables, $C_{Ass} \subseteq C$, the **dynamically controllable envelope** of an unassigned variable, $c \in (C - C_{Ass})$, is the set of prehistories of c for which there exists a viable dynamic execution strategy.

In other words, the envelope of a decision, given a partial assignment, is the subset of possible outcomes of earlier contingent links for which a viable strategy exists. It is similar to the notion of a relaxation of an over-constrained CCTPU, which also allows tightening contingent links, but captures all ways of making the subproblem dynamically controllable. A detailed example of a DC envelope, for the problem in Figure 4.13, is given by Equation (4.14) at the end of Section 4.4.4, after introducing the key processes of the approach.

The DC envelope of a set of discrete variables is a combination of the variables' envelopes. For a contingent link in the prehistory of two or more variables, all conditions on the link apply, so that the envelope is the intersection. Where two variables have different links in their prehistory, the envelope is defined over the union of their prehistories.

A CCTPU is dynamically controllable if the DC envelope of a partial assignment covers all possible outcomes of uncertainties in the problem. It means the partial assignment can be made statically, and there exists a dynamic strategy for the future (discrete and scheduling) choices. Hence, our approach (1) builds a search tree by expanding on variables, (2) extracts the dynamically controllable envelopes of STPUs at leaves and (3) aggregates those envelopes as the dynamically controllable envelopes of non-leaf nodes. If the DC envelope of any node covers all uncertainty, a dynamic execution strategy can be extracted from this node.

Next, we describe the general idea of the approach; the details are introduced in the following four subsections.

4.4.1 Algorithm Structure

Our dynamic controllability checking algorithm for a CCTPU (Algorithm 10) is a recursive tree search. Each leaf node is the STPU obtained from a full assignment to discrete variables, while interior nodes are CCTPUs with partial assignments. The root is the original CCTPU. Every other node has one parent that eliminates the assignment to the "latest" variable. The chronological order of variables is defined in the next subsection.

The algorithm traverses the tree depth-first, assigning variables in chronological order. *NextUnassignedVariable* (line 1) returns the next variable to be assigned. If every variable has been assigned, the current node is a leaf (lines 2 – 8); in this case, we extract the conflict resolution constraints that must be satisfied to make the leaf dynamically controllable and record those in $Node.S$ as its DC envelope (line 3). The detailed description of *DCEnvelopeSTPU* is in section 4.3. A non-leaf node (lines 9 –18) is expanded by assigning values to the next variable and exploring those nodes recursively. After the child branches of a node have been investigated, their DC envelopes are combined and recorded as the DC envelope of the current node (line 14). If the envelope of a node is dynamically controllable, then so is the CCTPU and the algorithm returns successfully.

The following subsections explain the three subroutines besides *DCEnvelopeSTPU*: *NextUnassignedVariables*, which determines the order to assign discrete variables, *Union* (line 14), which combines envelopes from branches of the same node, and *isDC* (line 15), which checks if the current solution includes a dynamic strategy for the original problem. We explain these procedures in the following subsections.

Algorithm 10: Checking dynamic controllability of a CCTPU.

Input: A $Node =< N, A, S >$ includes a CCTPU N, a vector of assignments A and the solution of the current node S.
Output: TRUE/FALSE
Algorithm: TreeSearch($Node$)

```
1  c ← NextUnassignedVariable(Node)
2  if isNull(c) then
3  |  Node.S ←DCEnvelopeSTPU(Node)
4  |  if isDC(Node.S) then
5  |  |  return TRUE
6  |  endif
7  |  return FALSE
8  endif
9  for dc_i in D(c) do
10 |  NewNode ← Assign(Node, c, dc_i)
11 |  if TreeSearch(NewNode) then
12 |  |  return TRUE
13 |  endif
14 |  Node.S ← Union(Node.S, NewNode.S)
15 |  if isDC(Node.S) then
16 |  |  return TRUE
17 |  endif
18 end
19 return FALSE
```

4.4.2 Branching Rule

The order in which the algorithm assigns variables obeys the execution process. During execution, a decision can observe and depend on previous assignments. Even

when some variables are decided in parallel, the branching rule assigns them in a sequence that respects to this order.

If a variable's decision time point is after any links that may be activated by another discrete variable, there is a **dependency** from the earlier to the later variable. For example, in Figure 4.13 the choice between $c_2 = 1$ and $c_2 = 2$ depends on the choice for c_1. Dependencies among discrete variables form a directed graph.

To find the chronological order, we build dependencies among variables. Recall our definition of precedences (Definitions 4.7 and 4.8), we give the following definition.

Definition 4.15. If a link $e \preceq DT(c_i)$ and e can be activated or deactivated by c_j, then c_i **depends on** c_j, denoted as $c_i \prec c_j$. If a link $e \preceq_{PA} DT(c_i)$ and e can be deactivated (the same as activated) by $A(c_j)$, then c_i **depends on** c_j under partial assignment PA, denoted as $c_i \prec_{PA} c_j$.

If the dependencies among variables do not cause a cycle, we can use topological sort to get all possible chronological orders. Otherwise, the dependency cycles can be treated as conflicts. If the network is dynamically controllable, at least one of the links within a cycle need to be deactivated by assignments. Therefore, we can just try different possible orders, if a feasible dynamic strategy is available under one of such orders, the network is still dynamically controllable.

With these definitions, we can formulate a dependency graph that enables to get chronological orders to assign variables. However, it may have cyclic dependencies, as shown in Figures 4.2 and 4.10 (the diamond nodes are decision timepoints), which will cause a failure to select the next unassigned variable. The cyclic dependencies containing links with different assignments from the same variable, like Figure 4.2, are ruled out by our assumption 4.1 because the decision time $DT(c_3)$ has to be placed before node A, B, C and D which breaks the dependencies $c_1 \prec c_3$ and $c_2 \prec c_3$. Therefore, assigning c_3 before entering the cycle will break the cyclic dependency. However, cyclic dependencies that contain links with labels of different variables as in Figure 4.10 cannot be solved in the same way. To deal with this case, we can remodel the problem by adding a discrete variable c_0, attaching assignments to each link included in the cyclic dependency in Figure 4.11, so that the cyclic dependency will be broken by any assignment of c_0 which is made before assigning c_1, c_2 and c_3.

Because all cyclic dependencies either contain links with labels of repeated variables or different variables, which can be broken by either assumption 4.1 or remodelling, we can always find the next unassigned variable that does not depend on any other unassigned variables. The order to assign variables is the chronological order we use to branch the search tree.

4.4.3 Combining DC Envelopes

The combining process aims to answer under which condition an assignment of the current discrete variable, c, can be made at its decision time point $DT(c)$ such that

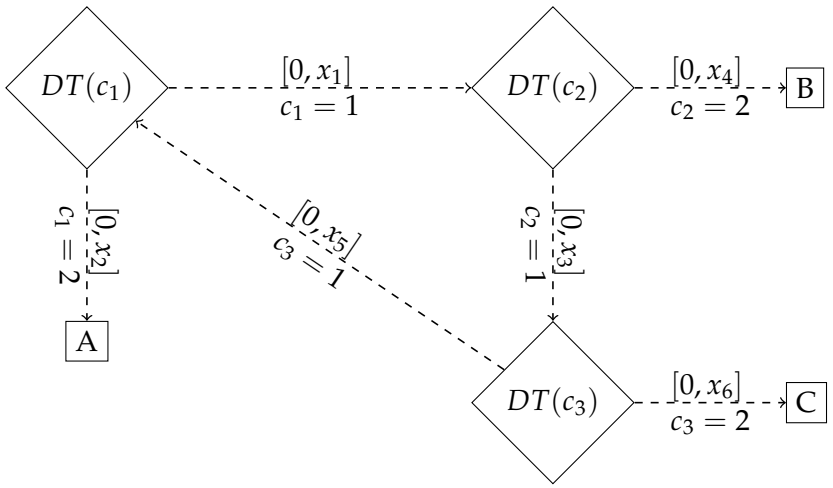


Figure 4.10: Cyclic Dependencies among Variables II

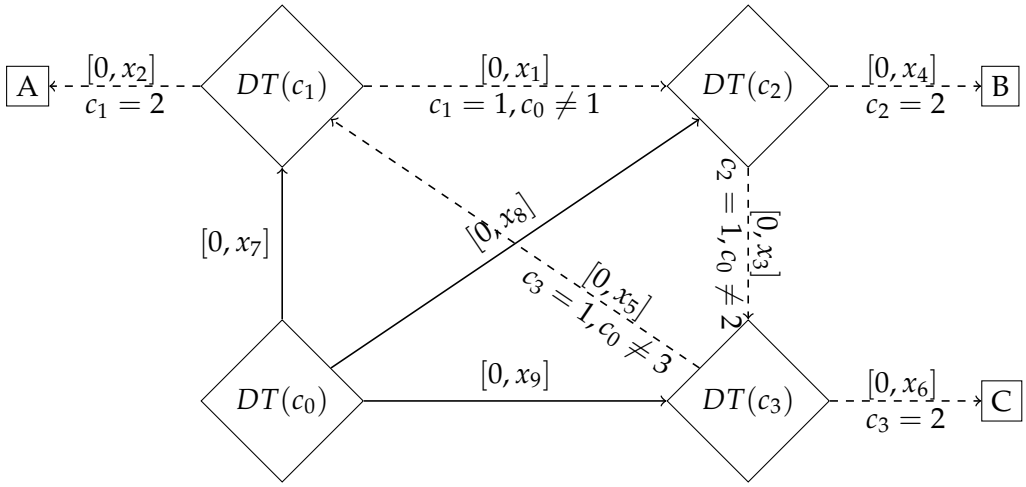


Figure 4.11: Remodelling Cyclic Dependencies among Variables II

the future part of the problem is dynamically controllable. This condition is the DC envelope of the current node.

Ideally, dynamic choices should be based on the whole prehistory before $DT(c)$. Conflicts with different paths have different prehistories before $DT(c)$. Recall our Assumption 4.3, that $DT(c)$ is the end point of a contingent link. Since no more than one contingent link can finish at a node, this means we only consider one contingent link (or the sum of a sequence of contingent inks) as the latest observation in each combining process.

In combining envelopes, we may split observable uncertainty, which is before $DT(c)$, so that each child branch only tackles part of it. It may resolve conflicts, or at least relax them, meaning less strict constraints on the prehistory may replace the

original conflict resolution. Therefore, a conflict can be resolved through several combining processes, and each combining process aggregates child envelopes according to Equation (4.2). The dynamically controllable envelope as defined in Formula 4.2 helps us understand the combining process, but the elements of the dynamically controllable envelope are uncountable since the domain of bounds is continuous. Therefore, we use the constraints of conflict resolutions to represent dynamically controllable envelopes, whose solution space is the set of relaxations according to Definition 4.13.

The envelope of the node to assign discrete variable c under partial assignment PA is a disjunction of child nodes' envelopes:

$$\Phi = \bigvee_{dc_l \in D(c)} \bigwedge_{j \in Conf} \bigvee_{k \in res_j} \sum_{i' \in res_{jk} \cap ER} x'_{i'} \geq a_{jkl}, \tag{4.10}$$

where $ER = \{e_i | e_i \in EU \text{ such that } e_i \prec_{PA} DT(c)\}$ and $a_{jkl} = -\sum_{i \in res_{jk} \setminus ER} x_i$ is the sum of bounds of links after $DT(c)$. As the selection operator in equation (4.3) may fail because of no suitable element can be found, the child envelopes may be infeasible when reducing relaxable links. Those envelopes will be removed before being combined. Furthermore, if no child envelope is available to be combined, the current envelope does not contain dynamically controllable execution strategy, which does not need to be combined in upper levels.

The envelope can be expanded into a conjunction of disjunctions, as shown in the following equation.

$$\Phi = \bigwedge \bigvee \sum_{i' \in res_{jk} \cap ER} x'_{i'} \geq a_{jkl} \tag{4.11}$$

Each conjunct takes one conflict resolution (which is a disjunction of linear constraints) from every child node branch in equation (4.10), so the number of conjuncts can be the product of the number of constraints in each in-failed child node's envelope. Transforming the envelope into this form, same as that of equation (4.9), makes the combining process uniform at all levels of the tree. Φ can cover the whole uncertainty if and only its negation is infeasible within the original bounds of relaxable contingent links.

$$\begin{aligned} \neg\Phi &= \neg \bigwedge \bigvee \sum_{i' \in res_{jk} \cap ER} x'_{i'} \geq a_{jkl} \\ &= \bigvee (\neg \bigvee \sum_{i' \in res_{jk} \cap ER} x'_{i'} \geq a_{jkl}) \\ &= \bigvee \bigwedge \sum_{i' \in res_{jk} \cap ER} x'_{i'} < a_{jkl} \end{aligned} \tag{4.12}$$

As each disjunct in equation (4.12) contains a conjunction of linear constraints, we can use an LP solver to perform its test. A disjunct can be removed, when it is infeasible, which means its negation covers the whole uncertainty. We only keep

the rest disjunction for the following combining processes. Additionally, if every disjunct is infeasible, Φ covers every uncertain situation in prehistory, which means the current node is dynamically controllable and an execution strategy can be found within the combined envelope.

4.4.3.1 Decision Consistency in Prehistory

The decision consistency aims to explain why and how we consider one contingent link in each combining process.

During execution, the strategy splits at a decision timepoint $DT(c)$ by assigning variable c. For example in the illustrative Figure 4.1, after timepoint $E1'$, Mr. P only needs to consider one branch after choosing his dinner option. However, even though we keep all contingent links prior to the decision timepoint as relaxable links in the dynamically controllable envelopes, only one of them can be combined each time. As Figure 4.5 shows, there may exist more than one contingent links before the latest decision timepoint c. If one child branch tackles both $s \to t_1$ and $r_2 \to t_2$ partially, the temporal scheduling of r_1 and r_2 separate the two observations. The strategy is related to dynamic controllability reduction rules. The temporal scheduling of r_1 and r_2 only deal with the relaxed $s \to t_1$.

When combining envelopes from branching decisions of a discrete variable, the envelopes must contain relaxable variables on no more than one contingent link. If the envelope contains linear constraints on variables of both contingent links in Figure 4.5, deciding requirement links between two partial observations means the decision has to be made before the observation of the second contingent link, which is not dynamically controllable. Otherwise, the envelope contains separate constraints on both contingent links. Although several such envelopes can combine and cover the whole prehistory, the decision on the discrete variable has been partially made in the decision of the requirement link in this case, which breaks our assumption 4.1.

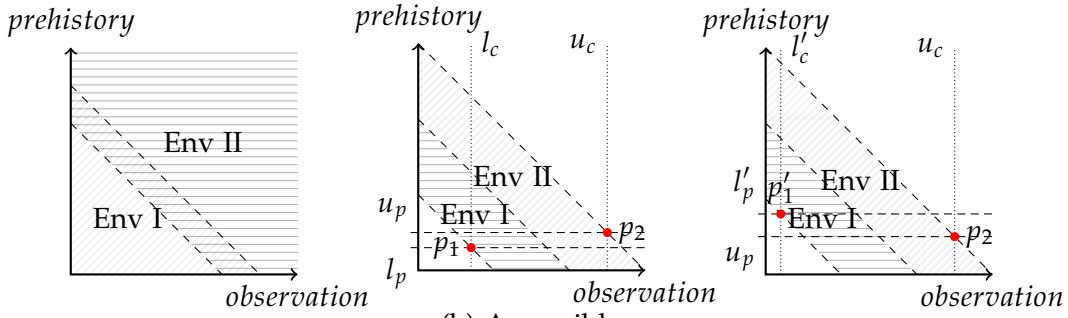
In each combining process, we consider one contingent link as the critical observation that can be partially tackled in child branches by making a choice for one discrete variable. We try the end node of every contingent link that precedes the latest decision time of c as $DT(c)$, replacing the variables representing its lower and upper bounds by a single variable x_{ij}. Then, the envelope answers under which condition for all $x_{ij} \in [l^c_{ij}, u^c_{ij}]$ there exists a choice dc_i such that x_{ij} satisfies constraints in the envelope of the child node with $c = dc_i$. After selecting the key observation, we check the feasibility of each disjunct in equation (4.12) with the following linear programming problem.

$$s.t. \bigwedge \sum_{i' \in res_{jk} \cap ER \setminus \{x_{ij}\}} p_{i'} + x_{ij} < a_{jkl} \tag{4.13}$$
$$l^c_{ij} \leq x_{ij} \leq u^c_{ij}$$

Besides the change of x_{ij}, we also temporally use $p_{i'}$ to represent the other edges in the conflict resolutions. It is because both lower and upper bounds of contingent links

in the envelope represent the uncertain projection in prehistory, which is observed at the time the choice will be made.

Figure 4.12 illustrates situations of the reduction of a disjunction of constraints with a common contingent link. Figure 4.12(a) shows an easy situation, where $E_c^I = \{[-Inf, a]\}$, $E_c^{II} = \{[b, Inf]\}$ and $a \geq b$, so the union of envelopes I and II covers the whole space. It means that for all situations in the prehistory, no matter what is observed, there will always be a feasible option that leads to a dynamic strategy. Figure 4.12(b) illustrates a general and feasible situation. Envelopes I and II cover the area from the lower bound of E_c^I to the upper bound of E_c^{II}. The intersections of the observation bounds $[l_c, u_c]$ with the edges of Envelopes I and II are p_1 and p_2. The combined envelope indicates that if the prehistory is within $[l_p, u_p]$, there will be a feasible choice for c. Figure 4.12(c) illustrates an infeasible situation. The uncertainty is so significant that the combined envelope cannot provide a viable range for the prehistory before the observation.



(a) An feasible example (b) A possibly feasible example (c) An infeasible example

Figure 4.12: Combined Envelope

Following our assumptions, the DC envelope (1) is represented by constraints on bounds of links prior to the decision time point of the variable, (2) enables different assignments to the variable, which also implies different following schedules, and (3) contains a dynamic strategy over the prehistory that is consistent for different assignments.

4.4.4 DC Checking for the Combined Envelope

This subsection aims to answer what kind of combined envelopes means the problem is solved or unsolvable.

A node is dynamically controllable if its DC envelope covers all uncertain situations implied by the problem. If the negation of an envelope is infeasible within the original bounds of the problem, which means every uncertain situation can be solved within one disjunctive branch, the problem is dynamically controllable. The envelope is disjunctive, and a separate check is done for each disjunct.

If the problem is not solved during the combining process at any interior nodes of the search tree and the DC envelope of the root is still not dynamically controllable, then a dynamic strategy satisfying our assumptions does not exist.

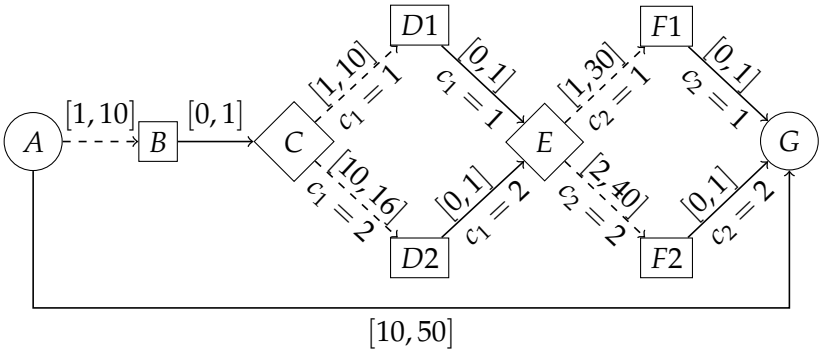


Figure 4.13: An example of CCTPU with two discrete variables

We explain the combining process and node checking with an example with two discrete variables, shown in Figure 4.13. The circles are controllable nodes, the squares are uncontrollable nodes and the diamond nodes are latest decision timepoints. The dashed lines are contingent links and the solid lines are requirement links. There two decision variables c_1 and c_2 and each of them has two options.

This problem cannot be solved with a fixed assignment because the STPU in each leaf node contains conflicts.

The algorithm first arrives at the leaf $\{c_1 = 1, c_2 = 1\}$ and extracts the conflict

$$A \xrightarrow{b:1} B \xrightarrow{1} C \xrightarrow{d_1:1} D_1 \xrightarrow{1} E \xrightarrow{f_1:1} F_1 \xrightarrow{1} G \xrightarrow{-10} A$$

Then in leaf $\{c_1 = 1, c_2 = 2\}$, the conflicts are

$$(1) \quad A \xleftarrow{B:-10} B \xleftarrow{0} C \xleftarrow{D_1:-10} D_1 \xleftarrow{0} E \xleftarrow{F_2:-40} F_2 \xleftarrow{0} G \xleftarrow{50} A$$
$$(2) \quad A \xrightarrow{b:1} B \xrightarrow{1} C \xrightarrow{d_1:1} D_1 \xrightarrow{1} E \xrightarrow{f_2:2} F_2 \xrightarrow{1} G \xrightarrow{-10} A$$

Potential $DT(c_2)$ is B or D_1 (due to Assumption 4.3). Because making decision of c_2 at B will never satisfy the branch $\{c_1 = 1, c_2 = 2\}$, we illustrate the solution of making decision at D_1. The envelope at node $\{c_1 = 1\}$ is the solution space of the following constraints:

$$\vee \begin{cases} l'_{AB} + u'_{BC} + l'_{CD_1} + u_{D_1E} + l_{EF_1} + u_{F_1G} - l_{AG} \geq 0 \\ -u'_{AB} - l'_{BC} - u'_{CD_1} - l_{D_1E} - u_{EF_2} - l_{F_2G} + u_{AG} \geq 0 \\ l'_{AB} + u'_{BC} + l'_{CD_1} + u_{D_1E} + l_{EF_2} + u_{F_2G} - l_{AG} \geq 0 \end{cases} \tag{4.14}$$

where the variables x' are links before the decision timepoint. With variables x representing observations and p representing preconditions, the constraints are expanded

as

$$\wedge \begin{cases} & p'_{AB} + p'_{BC} + x'_{CD_1} \geq 7 \quad (c_2 = 1) \\ \vee & p'_{AB} + p'_{BC} + x'_{CD_1} \leq 10 \quad (c_2 = 2) \end{cases} \wedge \begin{cases} & p'_{AB} + p'_{BC} + x'_{CD_1} \geq 7 \quad (c_2 = 1) \\ \vee & p'_{AB} + p'_{BC} + x'_{CD_1} \geq 6 \quad (c_2 = 2) \end{cases} \tag{4.15}$$

The first branch in equation 4.15 can be cut because its negation causes infeasibility. The other branch's negation is still feasible, which means it does not cover all uncertainty in its prehistory. Recovering variables to their original bounds, the left constraint $l'_{AB} + u'_{BC} + l_{CD_1} \geq 6$ must be kept to the next combining process. In the next step, our algorithm explores node $\{c_1 = 2\}$. Using the same process, the envelope of node $\{c_1 = 2\}$ is $\{u'_{AB} + l'_{BC} + u_{CD_2} \leq 20\}$. The decision time point is $DT(c_1) = B$, the observation is $A \xrightarrow{[1,10]} B$ and the prehistory before the observation is empty. The combining process results

$$\begin{array}{rll} & p_\emptyset + x_{AB} & \geq 4 \quad (c_1 = 1) \\ \vee & p_\emptyset + x_{AB} & \leq 4 \quad (c_1 = 2) \end{array}$$

whose negation is infeasible. The problem is dynamically controllable.

The dynamic strategy is: (1) if $p_{AB} \geq 4$ at B, then make assignment $c_1 = 1$, otherwise $c_1 = 2$; (2) after choosing $c_1 = 1$, C is scheduled based on p_{AB}, so that AD_1 will always be longer than 6. At D_1 if $p_{AD_1} \geq 7$, then choose $c_2 = 1$, if $p_{AD_1} \in [6, 10]$, choose $c_2 = 2$, (3) after choosing $c_1 = 2$, C is scheduled immediately after B, and $c_2 = 1$ at D_2, (4) the scheduling of other controllable time points are inferred by the reduction rules of dynamic control.

4.5 Approach Validation

If the algorithm finds a node whose envelope passes the DC check, then the CCTPU is dynamically controllable. The strategy is to make choices according to the partial assignment statically and following the observation of one of the contingent links for the remaining variables. It is valid within the bounds of the prehistory, which are verified by the final DC check to contain all potential outcomes uncertain links. However, the algorithm is not complete, in the sense that it will find only strategies that meet Assumptions 4.2 and 4.3.

In this section, we prove that (1) given an STPU, the conflicts extracted by algorithm 7 is a complete set of conflicts; (2) given a CCTPU, the solution of our method has a dynamically controllable execution strategy with the dynamic assignment on discrete variables satisfying assumptions we have made. By proving (1), we show the correctness and completeness of extracting DC envelopes of STPU.

4.5.1 Validation of Dynamically Controllable Envelopes of STPU

In section 4.3, the modified dynamic controllability checking algorithm extracts a complete set of conflicts of a given STPU. Using disjunctive linear constraints as conflict resolutions, we represent the dynamically controllable envelopes of STPU by the solution space of the constraint model. To validate the correctness and completeness of our method, we prove that (1) all solutions satisfying constraints of their dynamically controllable envelope are dynamically controllable; (2) a dynamically controllable STPU satisfies its dynamically controllable envelope constraints.

In order to prove correctness, we introduce the following theorem.

Theorem 4.1. *If in a conflict, one negative link has more than one occurrences and one occurrence can propagate to another occurrence via a negative path, this conflict resolution can be the sum of other conflict resolutions.*

Proof. We use $\{e_1^-, e_2^-, ..., e_n^-, \}$ to represent the existence of negative link e^- in conflict C. e_i^- can propagate to e_j^- via a negative path in C. The distance of the negative path on C is $D_C(start(e_i^-), start(e_j^-))$ which is negative. Because $start(e_i^-)$ and $start(e_j^-)$ are the same node, the negative path is a negative cycle which can be resolved by a conflict resolution. If the rest part of C is negative, it is another conflict $D_C(start(e_j^-), start(e_i^-)) < 0$. Otherwise, C can be resolved by resolving $D_C(start(e_i^-), start(e_j^-))$. □

Thus, multiple existences of a negative link in a conflict must be part of added links not the start of any negative sequences. In other words, those multiple existences are not in the piled-up *backpropagation* processes (of Morris's cubic algorithm and our algorithm). Their *backpropagation* has been completed before finding the negative cycle. With Theorem 4.1, our algorithm only has to propagate one negative link once in a recursive process.

To prove the correctness, we only need to prove we found all conflicts that don't have multiple repetitions of a single negative link or the distances among parts of repetitions are non-negative. We also use Theorems 2.2 and 2.1 proved by [Morris, 2006, 2014] to prove the following theorem.

Theorem 4.2. ***Correctness:*** *The solutions satisfying constraints of conflict resolutions extracted by the modified dynamic controllability checking method (Algorithm 7) are dynamically controllable.*

Proof. To prove the correctness, we suppose conversely there is a solution N satisfying constraints of conflict resolutions extracted by the modified DC checking method but not dynamically controllable. According to Theorem 2.1 N has a semi-reducible negative cycle C. By Theorem 2.2, C can be found by DCbackprop procedure in the Morris's DC checking algorithm.

Based on the process of DCbackprop procedure, C results from a series of calling DCbackprop on negative edges $\{e_1^-, e_2^-, ..., e_k^-\}$ and each propagating backwards

through positive edges. These positive edges are either original or added edges (the added edges are positive paths in DCbackprop procedure, see line 9 in Algorithm 9).

First, we prove the case that all edges in C are original edges. The structure of C is shown in figure 4.14, where $\{p_1^+, p_2^+, ..., p_k^+\}$ are the paths of positive links propagated through, $\sum_i (e_i^- + p_i^+) < 0$ and $e_i^- + p_i^+ < 0$ for all i. Calling backpropagation from one negative edge e_i^- in C, all negative paths starting from e_i^- including p_i^+ in C will be enumerated by the DFS process until meeting other negative edges (line 17 in Algorithm 9) including the next negative edge e_{i+1}^-. Another backpropagation process will be called when meeting e_{i+1}^-. The recursive calls result in a termination of calling an ancestor negative link e_i^-. Therefore, C will be found. It contradicts that N satisfying the constraint of all conflict resolutions.

Then we consider the situation where C consists of original and added edges. In this situation shown in Figure 4.15, where e'^+_{add} is the added edge, which split p_i^+ into two parts, from the propagation of e_x^- through p_x^+ and $e_x^- + p_x^+ > 0$, the rest edges are kept as before. In this case, we have to prove that e'^+_{add} is added while propagating e_i^- (line 8 in algorithm 9)to the end point of e_x^-. There are four situations when calling backpropagation on e_x^- in the backpropagation of e_i^-: (1) it is an ancestor call, (2) the backpropagation of e_x^- is successfully terminated before, (3) the backpropagation has not been called or (4) it has been unsuccessfully called. In situation (1), C is a conflict that contains a negative path between two occurrences of e_x^-, so it can be resolved by other found conflict resolutions as shown in Theorem 4.1. In situation (2), e'^+_{add} has been added. In situation (3) and (4), e'^+_{add} is going to be added unless, in p_x^+, there is a negative link that has been called in an open ancestor call, which makes C contain a negative path between two existences of that negative link.

Therefore, C will be found. It contradicts that N satisfying the constraint of conflict resolution of C. □

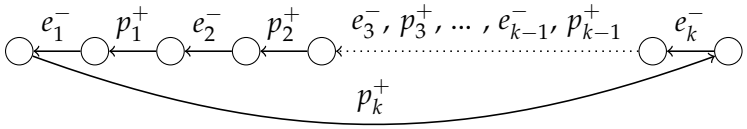


Figure 4.14: A semi-reducible negative cycle without added edges

Theorem 4.3. ***Completeness:*** *Every dynamically controllable STPU that is a relaxed instance (tightening contingent links) of the given STPU satisfies the constraints of conflict resolutions extracted by the modified dynamic controllability checking method (Algorithm 7).*

Proof. We assume there is a dynamically controllable STPU N that is a relaxed instance of the given STPU and N does not satisfy the constraints of conflict resolutions extracted by the modified DC checking method.

Thus, N violates a constraint CR_0 in the form of Equation 4.6, which means it violates every branch of the disjunctive linear constraint. However, the variables in

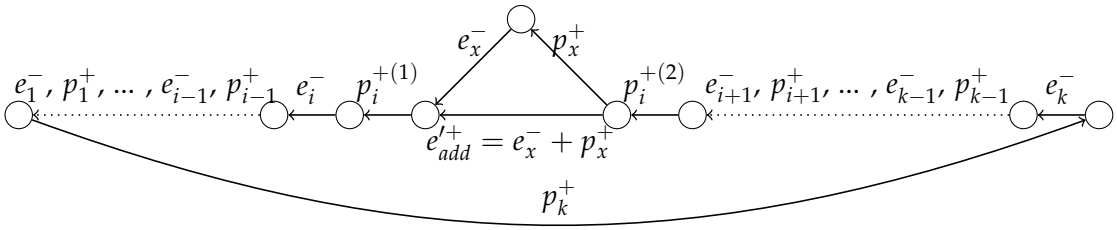


Figure 4.15: A semi-reducible negative cycle with added edges

CR_0 represent the bounds of links in conflict C_0. The dissatisfaction of N means the bounds of links of N formulate a conflict the same as C_0, which contradicts that N is dynamically controllable. □

4.5.2 Validation of Dynamic Controllability of CCTPU

In this subsection, we show that if a CCTPU is found to be dynamically controllable by our approach, there is a dynamic strategy that makes both temporal decisions and discrete variable assignments dynamically.

To validate our approach, we add the dynamic decisions of discrete variables to the validation of dynamic controllability of STPU by Morris and Muscettola [2000]. The dynamic decisions of discrete variables contain its decision timepoint $DT(c)$, which is the end of a contingent link or the start timepoint, and $A(c)$ based on the situation of the prehistory. If a CCTPU is dynamically controllable, it has a dynamically controllable envelope $Env(PA)$ that covers all uncertain situations in its prehistory. $Env(PA)$ is either in a leaf node when PA is A, or combined from child nodes that have one more assignment of a discrete variable c. If the algorithm finds a dynamically controllable STPU with a full assignment, its execution assigns all discrete variables at the beginning and executes the CCTPU as an STPU. Otherwise, a sequence of $DT(c)$ and $A(c)$ can be extracted from tracing back the combining process. Therefore, the execution is represented in Figure 4.16. The executions of discrete variables are added in the lines with star marks, the rest lines are the same as executing an STPU.

Hunsberger [2009] has introduced Real-time Execution Decision (RTED) to show actions that an agent may take while executing an STPU that is dynamically controllable. It can also be used to represent the execution strategy.

4.6 Experimental Results

We illustrate DC checking for CCTPU by comparing implementations of DC checking methods for the CCTPU with and without dynamic discrete choices. The DC checking method for the CCTPU which does not make assignments to discrete variables dynamically only considers scheduling time points dynamically. Our implementa-

Execute (CCTPU N)

0 Perform initial propagation from the start timepoint.

1* Assign discrete variables whose decision time has been reached, according to the prehistory and the envelope calculated. Immediately execute any executable timepoints that have reached their upper bounds.

2 Arbitrarily pick an executable timepoint TP that is live and enabled and not yet executed, and whose waits, if any, have all been satisfied.

3 Execute TP. Halt if network execution is complete. Otherwise, propagate the effect of the execution.

4* Advance current time, propagating the effect of any contingent timepoints that occur, until if the contingent timepoint is the decision timepoint of discrete variable c, go to 1*, or an executable timepoint becomes eligible for execution under 1* or 2.

5 Go to 1*.

Figure 4.16: Execution of the dynamic strategy of CCTPU

tion of this method checks every leaf node (full set of assignments) and considers the CCTPU dynamically controllable if there is one leaf that induces a dynamically controllable STPU.

4.6.1 Experimental Setup

We use the benchmark generator by Yu [2016] based on Zipcar problems [Yu and Williams, 2013; Yu et al., 2014]. Its application background is a car-sharing network. Each test case consists of missions with temporal requirements, each mission has a sequence of activities, and each activity can be done by choosing one option. An option contains controllable and uncontrollable links. All links are represented by their lower and upper bounds.

We use discrete variables to represent the choices for activities and attach assignments as labels to the links of each option. All temporal links are randomly generated except for the requirement on the overall duration of the missions, which randomly deviates by $\pm 20\%$ from the estimated bounds of the sequence of activities.

16000 test cases are ranging from 1–8 discrete variables with 1–10 options for each variable. Regarding the size of networks, the numbers of nodes, links and contingent links are varying from 11–170, 11–330 and 4–162, respectively.

4.6.2 Results

The result is shown in Figure 4.17. The tests are grouped by DC checking results in the chart. The white bars represent the result of implementation of a fixed assignment. 77.1% of the test cases are infeasible when using only fixed assignment. But just 22.4% are still infeasible while assigning discrete variables dynamically. The number of feasible results with fixed assignment is slightly fewer with the implementation of dynamic assignment because a combined solution can be found earlier in some test cases which contains a dynamically controllable STPU with a fixed assignment.

Figure 4.18 shows the runtime comparison between the implementations of dynamic controllability with fixed options and dynamic options. The y-axis describes the number of problems solved, where *solved* means the implementation terminates with a checking result of dynamically controllable or not. The runtime difference between two implementations is not obvious. It shows that making assignment dynamically does not cost much extra runtime. Furthermore, the implementation with dynamic assignment solves slightly more problems in time limitation above 1 second, which can be inferred that test cases that have a dynamic strategy with dynamic assignment terminate before exploring the whole search tree.

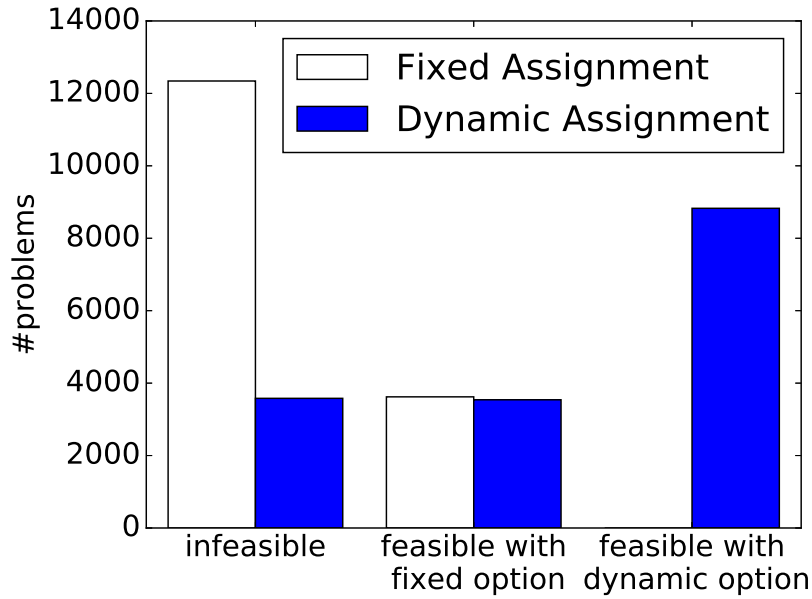


Figure 4.17: The distribution of results with fixed or dynamic options

4.6.3 A Simple Optimisation Experiment

To further demonstrate the advantage of making assignments dynamically, we make a simple optimisation experiment is based on the checking algorithm. The optimi-

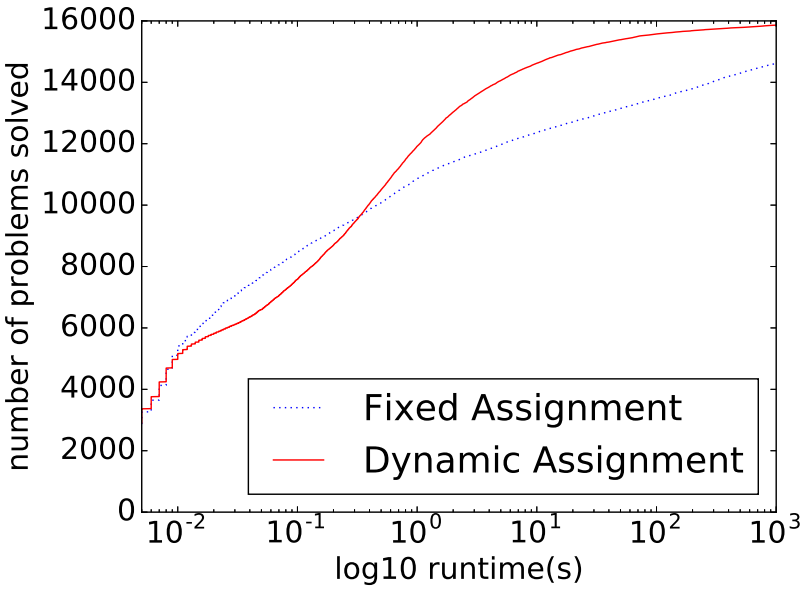


Figure 4.18: The number of problems solved within the time limitation.

sation experiment based on a robustness measure called maximum deviation introduced in Chapter 3. It maximises the minimum deviations (delays) of the uncertain durations, under which the problem is still dynamically controllable. It is a worst-case measure, so after setting the maximum deviation, a CCTPU is settled. Thus, we can use the algorithm introduced in this chapter to test its dynamic controllability.

In this experiment, we set the lower bounds of the contingent links as their original lower bounds and upper bounds are the sum of their lower bounds and the maximum deviation. The maximum deviation is set by a binary search above the checking algorithm.

In the data set with one discrete variable, which contain 2000 test cases, the result is shown in Figure 4.19. Less than half of the test cases remain in the same worst case if we use dynamic assignments instead of fixed assignments. 171, 302, 336 and 285 test cases have improvements of 10%, 20%, 30% and more than 30%, respectively.

4.7 Conclusion and Future Work

In this chapter, we extend dynamic controllability to CCTPU with making the assignment to discrete variables dynamically. Comparing to the previous work of making assignment statically, some test cases in the current CCTPU benchmarks are dynamically controllable with dynamic decisions on discrete variables but not dynamically controllable with a fixed assignment.

Assumptions 4.2 and 4.3 are essentially restrictions on the dynamic execution strategies that can be found by the algorithm we have presented in the next section. Removing or relaxing these assumptions is one of the future works. Removing them

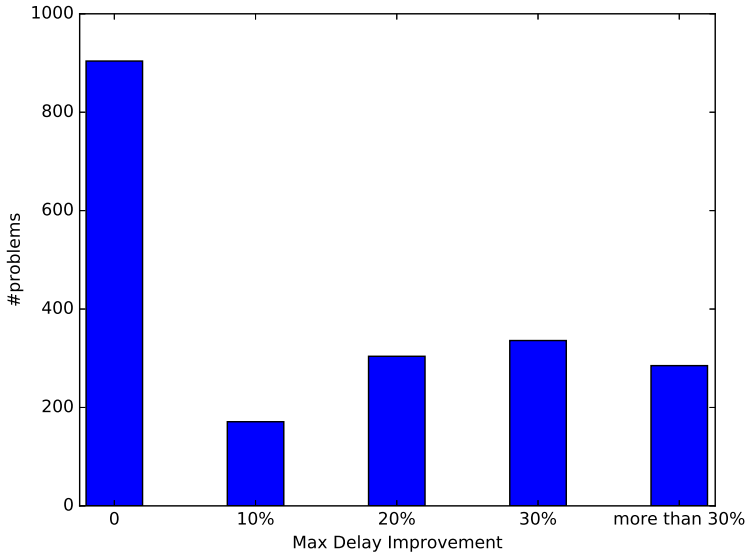


Figure 4.19: Improvement of Max Delay from Fixed Assignment to Dynamic Assignment

from the definition of dynamic controllability is straightforward (making the prehistory of a discrete variable $S\{\prec DT(c)\}$). Assumption 4.1 is not as easy. A dynamic strategy has to ensure that discrete choices are made so that there is no ambiguity about the activation of a link when the starting time point of the link is scheduled.

In future work, we will try to remove Assumptions 4.2 and 4.3, which limit the dynamic strategies that our algorithm can find. Another extension is to see how much improvement can be made in solving optimisation problems over a CCTPU when considering making choices dynamically.

Chapter 5

Optimising CCTPU

Adding controllable discrete variables makes dealing with the CCTPU more representative but more difficult than the STPU. Many real-world problems that can be represented by CCTPU are not controllable even with dynamic options because the temporal constraints are too tight or the uncertainty is too much. Instead of checking dynamic controllability, the work in this chapter aims to answer how controllable the problem is. On the other hand, solutions found without considering uncertainty may fail when executing in uncertain circumstances. The optimisation model in this section can also answer how robust or flexible these solutions can be when running with uncontrollable factors.

This chapter is an extension and combination of Chapters 3 and 4. The controllability constraint models constructed with variables for bounds of links inherits from Chapter 3. However, that constraint model can only deal with STPU that have no options. We extend the constraint model to the CCTPU to formulate constraints representing controllable options. The envelope-based dynamic controllability checking algorithm for the CCTPU, which has been discussed in Chapter 4, can only answer, given a CCTPU, whether it is dynamically controllable or not with dynamic options. In this chapter, we formulate a constraint model for the dynamic options, and a constraint model for the strategy with fixed options is provided as a comparison.

The strategy of a CCTPU consists of temporal decisions and variable assignments. In a feasible strategy of a CCTPU, both these arrangements can be made strongly controllable or dynamically controllable. A decision is strongly controllable if it can work without any observations. A decision is dynamically controllable if it varies by different observations of the past. Among the combinations of varying levels of controllability of these decisions, we formulate constraint models in the following settings: (1) the fully strong controllability that makes all decisions strongly controllable, (2) the mixed strong and dynamic controllability that makes temporal decisions dynamically controllable with fixed options on discrete variables and (3) the fully dynamic controllability that makes temporal decisions dynamically controllable with dynamic choices on discrete variables.

One of the challenges when formulating the optimisation model of dynamic controllability is the scalability. In the controllability constraint models of the CCTPU, the scalability problems exist in both formulating the constraints and solving the op-

timisation model. In this chapter, we propose a constraint model but leave how to solve the model efficiently as a future work.

In this chapter, we begin with examples to explain the motivation for the work (Section 5.1). In section 5.2 and 5.3, we introduce the optimisation model for the CCTPU with strong and dynamic controllability constraints and fixed and dynamic options. Section 5.4 presents the experimental results and section 5.5 is the conclusion and future work.

5.1 Problem Statement

In this section, we begin with illustrating the problem with motivating examples, then introduce the general model of the optimisation problem.

The first example is a modified CCTPU from Figure 4.1 that aims to show checking dynamic controllability is not enough to solve a CCTPU with preferences. The second example tries to answer how robust a given solution is by applying the robustness metric, introduced in Chapter 3, that measures the maximum deviation of a partial order schedule such that the schedule is still dynamically or strongly controllable.

5.1.1 Relaxing Over-constrained Problems

Checking dynamic controllability of temporal problems with uncertainty and controllable options cannot meet the demand for some applications, such as relaxing over-constrained problems. To illustrate this motivation, we modify the example in Figure 4.1 by adding reward values to each option and allowing an overlap of envelopes for two options (shown in Figure 5.1). If $c_1 = K$, the reward is 50; If $c_1 = S$, the reward is 100; The relaxation cost for each link is 10 per unit of relaxation. Two dynamic strategies with fixed assignment are (1) choosing $c_1 = K$ and relaxing $U_{E_1'X}$ by 5 to resolve the conflict raised by L_{E_1E}, the final preference is $50 - 5 \cdot 10 = 0$; (2) choosing $c_1 = S$ and relaxing L_{XE_3} and $L_{E_3'E}$ by 10 in total, the final preference is $100 - 10 \cdot 10 = 0$.

A dynamically controllable strategy with dynamic options to the example is that

- when the duration of the contingent event $E1 \rightarrow E1'$ is within $[35, 50]$, c_1 can be assigned with K and
- when it is within $[30, 40]$, c_1 can be assigned with S.

Thus, there is an overlapping interval $[35, 40]$ that can be dealt with both options. The user's preference that Mr. P likes the steak more than KFC cannot be considered in the dynamic controllability checking process.

To represent the favourite, we formulate an optimisation model, so that a new dynamic strategy with dynamic options under a specified objective function can be achieved. For example, in the problem of Figure 5.1, option $c_1 = S$ will be chosen for the overlapping interval $[35, 40]$.

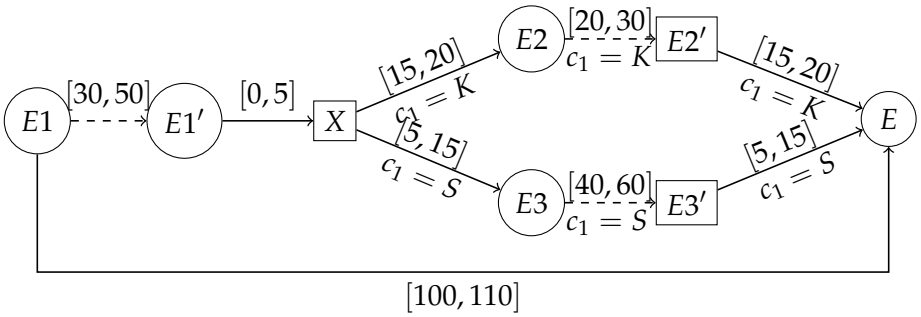


Figure 5.1: A modified example from Figure 4.1.

This example shows that the optimisation model helps to provide a superior solution rather than only a feasible solution.

5.1.2 Maximum Deviation of Partial Order Schedules

Different from the relaxation problems that require optimal strategies as solutions, many applications want to answer how good a solution is, like the robustness metrics we introduced in Chapter 3. The robustness metric Maximum Deviation, which calculates the maximum worst-case deviation that a partial order schedule can absorb yet keep its controllability, may help answer the question how robust the schedule is.

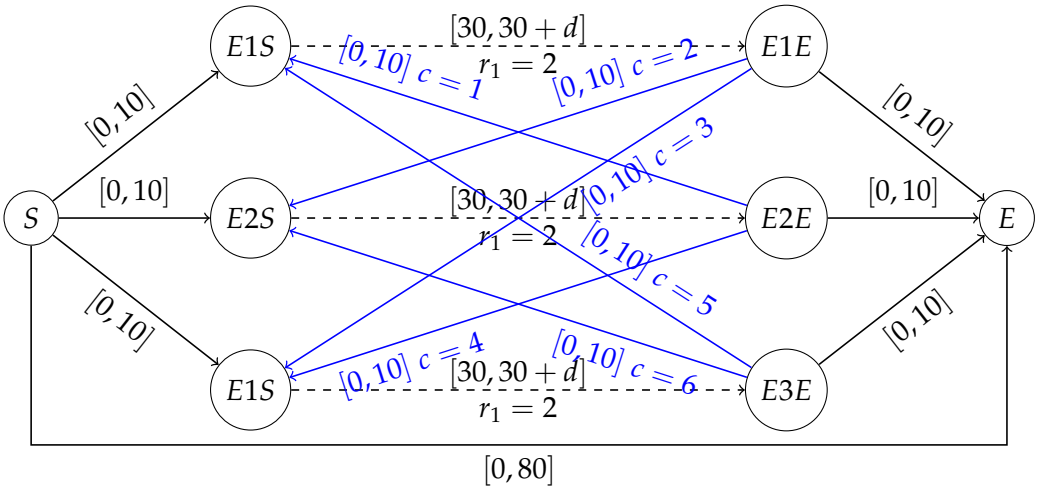


Figure 5.2: Using CCTPU to represent three POS of a problem

A schedule or temporal plan solved by a scheduler or planner can be represented by a consistent STN that does not have uncertainty. In Chapter 3, it is shown that modelling the links depicting the events that may contain disturbances during execution by contingent link enables an estimation of the robustness of a fixed schedule or temporal plan. In addition to that, in Partial Order Schedules, different ways to

add temporal constraints that resolve resource constraints can be switched by controllable discrete variables. For example in Figure 5.2, an RCPSP problem with three activities is shown. Each activity takes 2 units of resource I, and the total amount of the resource I is 5, which means the three activities cannot execute at the same time. Thus, 6 precedence links (blue lines) are added as partial order schedules resolving the resource conflicts. Formulating the constraint model of CCTPU under dynamic controllability with dynamic options enables solving the problem of optimising robustness and flexibility.

Therefore, the optimisation model of CCTPU can measure the robustness of partial order schedules and even give a more flexible solution than POS for scheduling problem.

5.1.3 General Formulation

The general form of the optimisation problem can be stated as: given a CCTPU, that is, the set of timepoints $V = V_E \cup V_U$, links $E = EC \cup EU$ and discrete variables C, their domain D and the labels of assignments attached to links, the problem is to set the bounds of the links and the assignments of the variables so as to optimise the objective function value (Equation 5.1).

$$\begin{aligned} opt \quad & f_{obj}(l_{ij}, u_{ij}, x | e_{ij} \in E) \\ s.t. \quad & N(l_{ij}, u_{ij}, x | e_{ij} \in E) \text{ is dynamically controllable or strongly controllable} \\ & \text{application-specific side constraints.} \end{aligned} \tag{5.1}$$

The decision variables are the lower and upper bounds of links and the auxiliary variables x representing the dynamic or fixed assignment. The solutions are constrained by the controllability constraints so that every solution satisfying the constraints has a feasible strategy. Other application-specific constraints are added depending on the problems.

5.2 Optimising CCTPU with Fixed Assignment

With the general optimisation model as Equation 5.1, we start with introducing the optimisation model of CCTPU with a fixed assignment. The optimisation model of CCTPU with dynamic assignment is introduced in the next section.

In the execution strategy of CCTPU with a fixed assignment, each discrete variable is assigned with one value from its domain. The links activated by the fixed assignment formulate an STPU. If the STPU is strongly or dynamically controllable, the CCTPU has strongly or dynamically controllable temporal strategy with the fixed assignment. Therefore, the optimisation model for the CCTPU with fixed assignment

can be formulated as shown in Equation (5.2).

$$\begin{aligned} opt \quad & f_{obj}(l_{ij}, u_{ij}, b | e_{ij} \in E, b \in A(C)) \\ s.t. \quad & \sum_{l}^{|D(c_k)|} b_{kl} = 1 \ \ \forall c_k \in C \\ & N'(l_{ij}, u_{ij} | e_{ij} \in E \wedge A(C) \models e_{ij}) \text{ is dynamically controllable} \\ & \text{application-specific side constraints} \end{aligned} \tag{5.2}$$

The decision variables l_{ij} and u_{ij}, represent the lower and upper bounds of link e_{ij}, and b_{kl} is a binary variable that when $b_{kl} = 1$ it represents the assignment $c_k = dc_l$. Constraint $\sum_{l}^{|D(c_k)|} b_{kl} = 1, \forall c_k \in C$ ensures that every variable is assigned by one value, which will be explained in subsection 5.2.1. The STPU N' activated by assignment $A(C)$, which is represented by binary variables $\{b_{kl} | c_k = dc_l, \forall c_k \in C\}$, is dynamically controllable.

Despite encoding the model into a MIP, this problem can also be solved by the Conflict-directed algorithm [Yu et al., 2014].

5.2.1 Activating Constraints with Fixed Assignment

In this subsection, we show how to formulate the constraints that activate links by the fixed assignment.

In a CCTPU, a link is activated if and only if its label does not have a conflict with the fixed assignment. For example, if the fixed assignment for a CCTPU contains $c_i = d$, the link that is activated should not have label $c_i = d'$ that $d' \neq d$. Thus, for each assignment $c = d_l$ in the label of link e_{ij}, we have that if $c \neq d_l$, e_{ij} is not activated, which is the same as e_{ij} not existing. We can express the deactivated link by setting their bounds unconstrained $l_{ij} \in [-\infty, \infty]$ and $u_{ij} \in [-\infty, \infty]$. On the other hand, when all labels attached to e_{ij} have been assigned by the fixed assignment A, the link is activated. If it is a requirement link with application- specific constraints $L_{ij} \leq l_{ij} \leq u_{ij} \leq U_{ij}$, then the constraints are activated by A. The same process can be applied to contingent links based on their application- specific constraints.

We use one binary variable $b_{kl} = 1$ to represent an assignment $c_k = d_l$, thus when a link e has a label with multiple assignments as $\ell_E(e) = \{c_{i_1} = d_{i_1 j_1}, c_{i_2} = d_{i_2 j_2} ... c_{i_k} = d_{i_k j_k}\}$, it is activated if and only if $b_{i_1 j_1}, b_{i_2 j_2} ... b_{i_k j_k}$ all equal to the size of the label $\|\ell_E(e)\|$. Therefore, the link e is activated iff

$$\sum_{c_k = d_l \in \ell_E(e)} b_{kl} = \|\ell_E(e)\|. \tag{5.3}$$

The disjunctive linear expression of activating link e_{ij} can be represented as fol-

lows:

$$\begin{aligned} \sum_{c_k=d_l\in\ell_E(e_{ij})} b_{kl} \leq N_\ell - 1 \quad \vee \quad L_{ij} \leq l_{ij} \leq u_{ij} \leq U_{ij}, \quad \forall e_{ij} \in EC \\ \sum_{c_k=d_l\in\ell_E(e_{ij})} b_{kl} \leq N_\ell - 1 \quad \vee \quad L_{ij} = l_{ij} \leq u_{ij} = U_{ij}, \quad \forall e_{ij} \in EU \end{aligned} \tag{5.4}$$

where N_ℓ is the size of the label of e_{ij}, lower-case l_{ij} and u_{ij} are the decision variables and upper-case L_{ij} and U_{ij} are the loose bounds.

If we let b represent the left-hand-side (LHS) of Equation (5.6), which is the switch of activating the link, equation (5.4) can be encoded in to a MIP (Equation 5.4').

$$\begin{cases} l_{ij} + (N_\ell - b)(M + L_{ij}) & \geq & L_{ij} \\ u_{ij} + (N_\ell - b)(-M + U_{ij}) & \leq & U_{ij} \\ l_{ij} - u_{ij} + (N_\ell - b)(-M) & \leq & 0 \end{cases} \quad \forall e_{ij} \in EC$$

$$\begin{cases} l_{ij} + (N_\ell - b)(M + L_{ij}) & \geq & L_{ij} \\ l_{ij} + (N_\ell - b)(-M + L_{ij}) & \leq & L_{ij} \\ u_{ij} + (N_\ell - b)(-M + U_{ij}) & \leq & U_{ij} \\ u_{ij} + (N_\ell - b)(M + U_{ij}) & \geq & U_{ij} \end{cases} \quad \forall e_{ij} \in EU \tag{5.4'}$$

where M is a big constant number that can be treated as infinity in the problem.

5.3 Optimising CCTPU with Dynamic Assignments

Based on the content in Chapter 3 and 4, the question how much flexibility, robustness or other objective values can be improved by making decisions of discrete variables dynamically may be answered. In this section, we try to formulate a constraint model to answer this question, using the dynamic controllability constraints introduced in Chapter 3 and the fully dynamic strategy defined in Chapter 4.

Based on the definition of fully dynamic controllability in Chapter 4, we propose an optimisation model of the CCTPU with dynamic assignments. The optimisation model is more complicated than the optimisation model with fixed options because of the following reasons: (1) the dynamic assignment formulates a dynamic strategy instead of a fixed assignment, which needs to be represented by decision variables, (2) a link activated by different options of a discrete variable may have different bounds in the branches serving various options, which must create separate decision variables to represent those bounds and (3) the way to model the partial uncertainty in the dynamically controllable envelopes that can be dealt in child branches is not clear.

5.3.1 Constraint Model Representing Dynamic Assignments

In order to model the dynamic assignment, decision variables representing the decision tree of the dynamic assignment and the decision timepoint for each assignment have to be introduced.

Different from the binary variables used in the constraint model in section 5.2,

which represent each option for each discrete variable, binary variables representing full assignments are added in the constraint model of dynamic assignment. For instance, binary variable $b_0 = 1$ stands for the full assignment $\{c_i = d_{i0} | \forall c_i \in C\}$ will be activated in the dynamic strategy. Therefore, the total number of binary variables representing full assignments is the product of the size of the variable domains.

Since a dynamic strategy aggregates different full assignments at decision timepoints of discrete variables, and we give a binary variable for each full assignment, if the decision time points are fixed, the constraint representing that a feasible dynamic strategy exist can be written as Equation (5.6)

$$\sum_{a \in A(C)} b_a \geq 1 \tag{5.6}$$

which means the strategy has at least one full assignment.

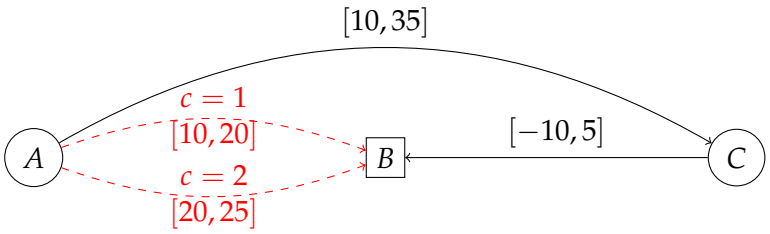
Although we can enumerate all possible choices of the decision timepoints and solve one constructed model for each setting, the constraint model in this chapter only considers fixed decision timepoints for the following reasons: (1) modelling flexible decision timepoints requires one integer variable to decide each combining point which is a repetition of a decision timepoint under a specific prehistory, and the domain of these integer variables are interdependent; (2) modelling flexible decision timepoints does not seem to be more flexible than setting fixed decision timepoints as the latest time. Therefore, in the constraint model, we give a fixed decision time point for each discrete variable, which is the end point of the last contingent link that is before every link that has a label that mentions the discrete variable.

5.3.2 Expanding the CCTPU

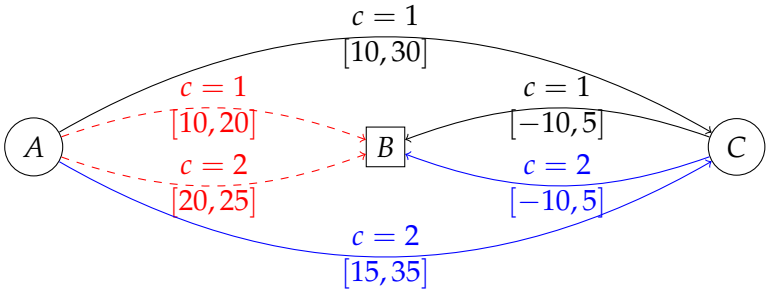
In this subsection, we aim to solve the second challenge mentioned at the beginning of the section, which is that different variables should be added to each link. We first illustrate the necessity of this process, then introduce how to expand the network by adding repetitions.

Expanding the CCTPU is necessary because a link in different branches may have different bounds under the controllability constraints. For example in Figure 5.3 (a), the dashed lines are contingent links. If the decision timepoint of discrete variable c is node A, the link AC and BC are not before the decision timepoint, so after making the assignment for c, we may have differently feasible bounds of these links according to the option of c. If $c = 1$, the duration of AB is within $[10, 20]$, based on triangular reduction $u_{AC} + l_{BC} \leq u_{AB}$, a feasible solution is to tighten u_{AC} by 5. In the same process, tightening l_{AC} by 5 is a feasible solution when $c = 2$. Thus, different bounds of link AC in branches $c = 1$ and $c = 2$ have to be attached to the branch label, although link AC does not have any label on the original problem. In this case, we add repetitions of link AC to the problem, which is shown in Figure 5.3 (b).

Algorithm 11 illustrates the expanding process. For every discrete variable, any links that are not before the decision timepoint of the discrete variable and do not



(a) a CCTPU problem



(b) the expanded CCTPU of the problem

Figure 5.3: An example illustrating the expanding process

have a related label of the variable are expanded into a few repetitions. The number of repetitions equals to the size of the domain of the discrete variable. Thus, if a link is expanded under several discrete variables, the number of repetitions will be the product of the size of the domains of the discrete variables, which means each partial assignment of those discrete variables has a repetition of the link.

Therefore, the activating constraints for each link are

$$\begin{aligned} \sum_{x \in X} b_x \geq N_X \quad \vee \quad L_{ij} \leq l_{ij} \leq u_{ij} \leq U_{ij}, \quad \forall e_{ij} \in EC \\ \sum_{x \in X} b_x \geq N_X \quad \vee \quad L_{ij} = l_{ij} \leq u_{ij} = U_{ij}, \quad \forall e_{ij} \in EU \end{aligned} \tag{5.7}$$

where $X \subseteq A$ is the subset of all full assignments A and every element x in X agrees with the label $\ell_E(e)$ of link e and N_X is the size of X. Equation (5.7) can be encoded into MIP in the same way as encoding Equation (5.4).

5.3.3 Constraints of Partial Uncertainty in DC Envelope

The third challenge of formulating the constraint model of CCTPU with dynamic options mentioned at the beginning of the section is the most important one since it can explain why the dynamic assignment enables the flexibility to deal with more uncertainty than the fixed assignment. In this subsection, we present a constraint model of the partial uncertainty in the prehistory that can be dealt by each branch.

The envelope-based dynamic controllability checking method for CCTPU in sec-

Algorithm 11: Expanding a given CCTPU before formulating constraint model with dynamic options.

Input: A CCTPU $N =< V, E, C, D, \ell_E >$ and the decision timepoints $DT(C)$
Output: An updated CCTPU N.
Algorithm: $ExpandCCTPU(N)$

```
1  for c ∈ C do
2      for e inE do
3          if e ⊀ DT(c) and ℓ_E(e) ∩ A(c) == ∅ then
4              for dc ∈ D(c) do
5                  e' = e;                          // create a new repetition of e
6                  ℓ_E(e').add(c = dc)
7                  E.add(e')
8              end
9              E.delete(e);                          // delete the expanded link e
10         endif
11     end
12 end
```

tion 4.4 uses the solution space of constraints to represent the dynamically controllable envelopes. Verifying the negation of the solution space is empty means a dynamically controllable envelope covers all uncertain situations in its prehistory. However, a dynamically controllable envelope contains a fully dynamic strategy that no matter what happened in the prehistory before the critical observation, there will always be a feasible allocation for the uncertainty of the critical observation such that each child branch can absorb the allocated uncertainty in the dynamically controllable envelope of the child branch. The feasible allocation may be a dynamic division of the key observation, which means the lower and upper bounds of the partial uncertainty dealt in each child branch are not fixed. Therefore, the dynamic allocation cannot be fitted into the optimisation model in Equation (5.1) that use lower and upper bounds as decision variables.

In this thesis, we introduce a constraint model that is more strict than the fully dynamic controllability but more flexible than the dynamic controllability with fixed assignments. In the model, we use a fixed allocation for each key observation, so that the partial uncertainties in the dynamically controllable envelope can be represented by contingent segments with fixed bounds. This model is "over-constrained" in that it does not allow for the full space of solutions which makes the objective value sub-optimal, because the solutions having fixed allocations of partial uncertainty have a fully dynamic strategy but the CCTPU may have a fully dynamic strategy that does not have fixed allocations of partial uncertainty.

The difference between the dynamic controllability checking algorithm that considers dynamic envelopes and the constraint model that considers fixed envelopes is illustrated in Figure 5.4. The figure shows a combining process of Envelope I and II that have overlaps, but example (b) has a prehistory with more uncertainty than example (a). Both examples are dynamically controllable by the dynamic controllability

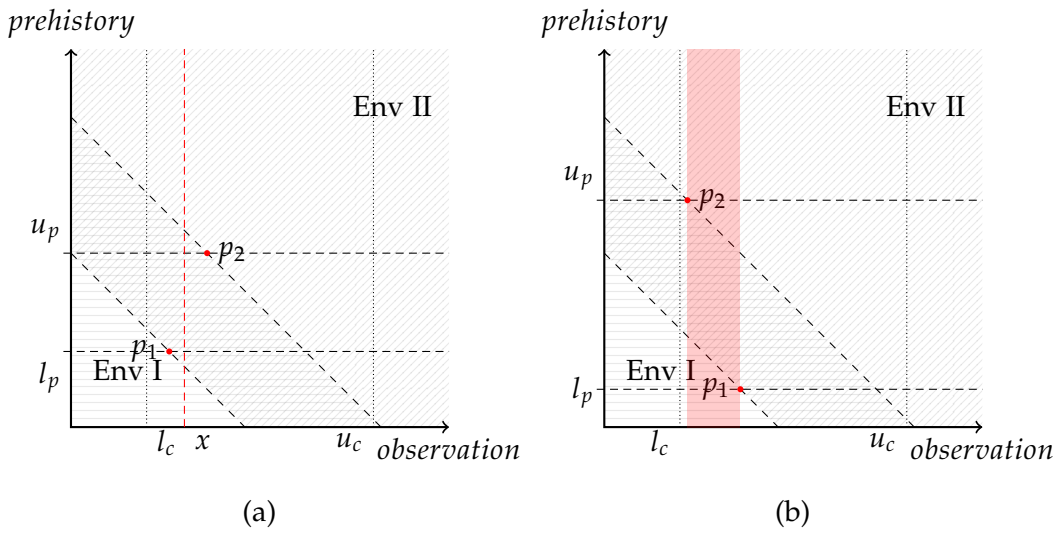


Figure 5.4: The difference between dynamic envelopes and fixed envelopes

checking algorithm in Chapter 4 because no matter what happened in the prehistory, there will always be a feasible way to allocate the key observable contingent link into the two envelopes of the child branches. We select p_1 which is the intersection of l_p and the bound of Envelope II and p_2 which is the intersection of u_p and the bound of Envelope I. These two points stand for the boundaries of the observation where each envelope can be selected. However, only example (a) has a fixed allocation that is shown by drawing a vertical line between p_1 and p_2 at x, $[l_c, x]$ is allocated to Envelope I and $[x, u_c]$ is allocated to Envelope II. On the other hand, example (b) does not have such a fixed allocation since p_2 is left to p_1 which means the uncertainty of the key observation between p_1 and p_2 (in the red shadow) has to be allocated dynamically based on the observation of the prehistory. Therefore, the constraint model in this chapter can represent the dynamic controllability of example (a) but cannot represent that for example (b). If modelling a relaxation problem with the dynamic strategy constrains the situation of example (a), the constraint model has to do relaxation to achieve a dynamically controllable solution with fixed allocations.

Next, we propose the constraint model that divide critical observations into fixed intervals that can be tackled in different successive branches. For each discrete variable c with domain $D(c)$, the key observation is the contingent link finishing at $DT(c)$. To the constraint model, $N_D + 1$ continuous variables $\{x_0, ..., x_{N_D}\}$ are added to represent N_D segments of the contingent link, where N_D is the size of the domain $D(c)$. If the key observable contingent link is activated, the activating constraints (Equation 5.7) can be used on x_0 and x_{N_D} as lower and upper bounds. Each child branch has to deal with one segment $[x_i, x_{i+1}]$ among them. We use a set of allocation constraints to restrict which segment will be selected by which child branch. A N_D^2

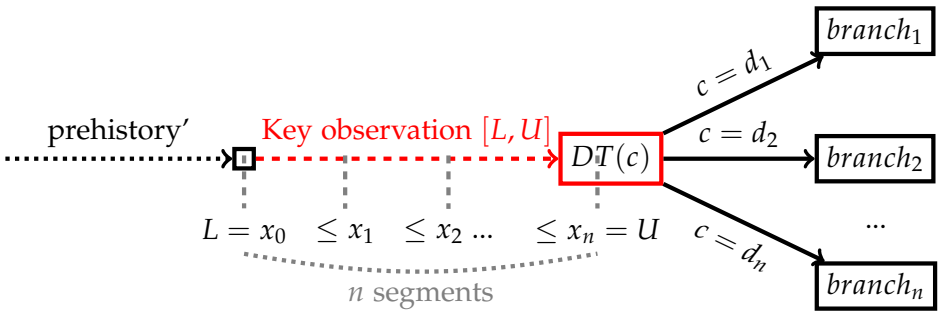


Figure 5.5: Illustration of the fixed division of key observations

matrix P of binary variables p_{ij} is introduced, where binary variables $p_{ij} = 1$ represents the i^{th} segment is taken by the j^{th} branch. The allocation constraints is shown in Equation (5.8).

$$\begin{cases} x_i & \leq x_{i+1} & & \forall i \in \{0, ..., N_D - 1\} \\ \sum_j p_{ij} & \leq 1 & & \forall i \in \{0, ..., N_D - 1\} \\ \sum_i p_{ij} & \leq 1 & & \forall j \in \{0, ..., N_D - 1\} \\ \sum_j p_{ij} & \leq 0 & \vee \quad x_i < x_{i+1} & \forall i \in \{0, ..., N_D - 1\} \\ p_{ij} & \leq 0 & \vee \quad (l_j \leq x_i \leq x_{i+1} \leq u_j) & \forall i, j \in \{0, ..., N_D - 1\} \end{cases} \tag{5.8}$$

The five equations mean: (1) $\{x_0, ..., x_{N_D}\}$ is in an increasing order; (2) Each segment is taken by no more than one child branch; (3) Each branch takes no more than one segment; (4) A non-empty segment will be taken by one child branch; (5) Binary variable $p_{ij} = 1$ represents the i^{th} segment is taken by the j^{th} branch. Last but not least, associating the allocation constraints to variables b_a of the dynamic strategy, Equation (5.9) means each branch taking a segment is activated branch,

$$\sum_i p_{ij} \leq 0 \vee \sum_{x \in X} b_x \leq N_X - 1, \forall j \in \{0, ..., N_D - 1\}, \tag{5.9}$$

where $X = \{a | a \in A, a \models \{c = d_j\}\}$ and N_X is the size of X. Therefore, every non-empty segment of the key observable contingent link is taken by at least one activated branch.

The solution of constraint model (Equation (5.6– Equation (5.9)) contains the dynamic strategy with fixed envelopes. For every $b_a = 0$, the branch activated by assignment a belongs to the strategy. Different activated branches are divided at $DT(c)$ according to the observation. Options depend on the duration of the observation and the fixed allocation of the envelopes.

5.4 Experimental Results

In this section, we encode the constraints models with fixed options and dynamic options into Mixed Integer Programs and solve them by GuRoBi to compare how much improvement can be made by making assignment dynamically. Additionally, the scalability of these two constraint models is tested.

The experimental setup uses the same application as Section 4.6.1. The test cases range from 1-4 discrete variables with 1-10 options for each variable.

5.4.1 Objective Function

In the experiment, we need to formulate a comparable objective function among different levels of controllability for CCTPU. For the application relaxing over-constrained CCTPU, we use

$$\min cost = \sum_{e_{ij} \in E} (\delta_{ij}^{l} + \delta_{ij}^{u}) \tag{5.10}$$

where δ_{ij}^{l} and δ_{ij}^{u} are the relaxations of the lower and upper bounds of relaxable links. We drop the reward function from the relaxation problem of CCTPU with fixed options since the solution of the constraint model with dynamic options will have a strategy instead of a fixed assignment, which makes the reward function not comparable. An alternative way to fix this problem is that we can make a reward function by using the minimal reward among fixed assignments of the strategy to illustrate the worst case.

$$\max obj = R - cost \tag{5.11}$$

The reward R satisfies the constraints

$$R \leq (1 - b_a) r_a + b_a M, \forall a \in A \tag{5.12}$$

where $r_a = \sum_{\{c=d\} \in a} reward_{c=d}$ is the reward of the full assignment a and M is a big number that can be set as the best reward among all assignments.

5.4.2 Results

We compare the relaxation cost given by Equation 5.10 among strong controllability, dynamic controllability with fixed options (Equation 5.2) and dynamic options (Equation 5.6 – Equation 5.9). The result is shown in Figure 5.6.

We classify the result of improvement from strong controllability to dynamic controllability with fixed options and dynamic controllability with fixed options to dynamic options into four sets: no increase, 10% improvement, 50% improvement and more than 50%, and count the number of test cases in each class. The total number of solved problems within a runtime limitation of 8000 seconds is 1271. The result

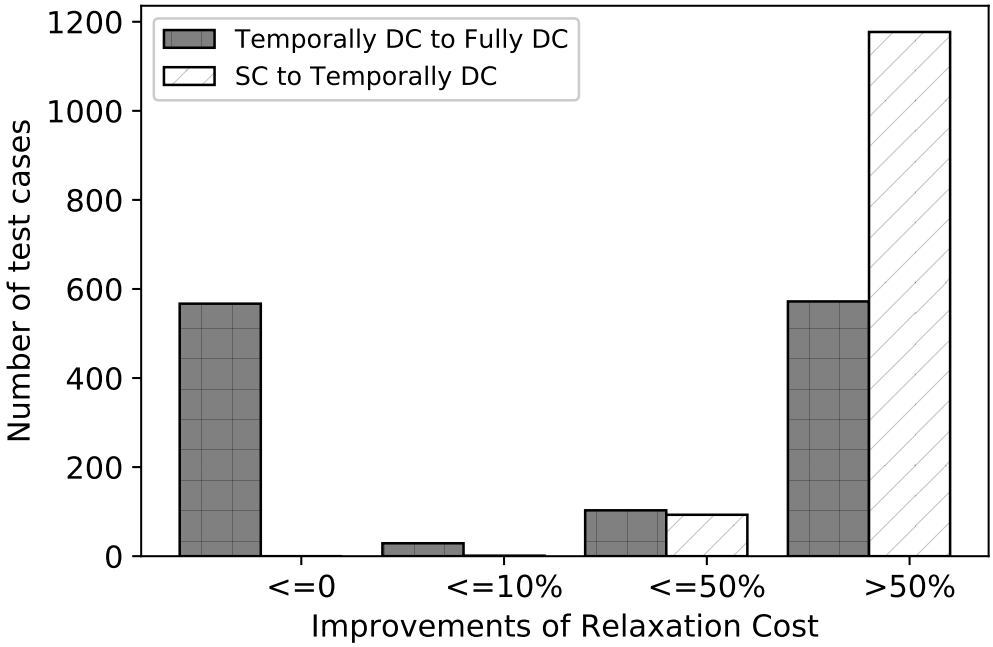


Figure 5.6: Improvements of Making Decisions Dynamically

shows that the number of test cases having more than 50% improvement from strong controllability to dynamic controllability with fixed options is 1177, which is 92.6% of the solved problems and the rest 7.4% test cases have improvements between 10% and 50%. The improvement of relaxation cost from dynamic controllability with fixed options to dynamic options is smaller but still apparent. 567 test cases have a dynamically controllable strategy with fixed options, which means their relaxation cost are zeros and cannot be improved any more. 29, 103 and 572 test cases have 10%, 50% and more than 50% improvements from dynamic controllability with fixed options to dynamic options, respectively.

Since the constraint model used in this chapter is sub-optimal, we use the implementation checking dynamic controllability in Chapter 4 to test the same data set. 591 test cases have no relaxation cost under the constraint representing dynamic controllability with fixed options but 741 test cases are dynamically controllable in the checking algorithm, which means 150 test cases are dynamically controllable with dynamic options and dynamic allocation of uncertain situations but have to be relaxed to satisfy the dynamic controllability constraints in this chapter.

We also test the scalability of the constraint models in different levels of controllability in runtime (Figure 5.7). Since the constraint model is a baseline method to solve the optimisation method, the performance is not unexpected.

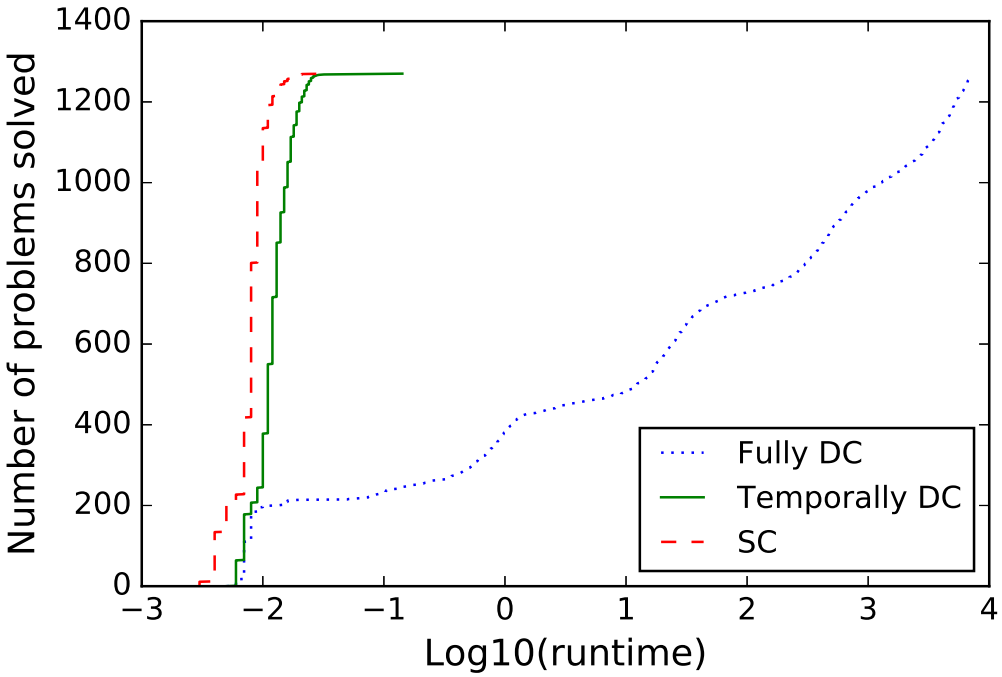


Figure 5.7: Runtime Comparison

5.5 Conclusion

With the constraint model of dynamic controllability for CCTPU with dynamic options, we can answer the question of how much improvement can be achieved by making controllable choices dynamically. The constraint model can also explain how flexible, robust or controllable a temporal problem with uncertainty and controllable options is.

Furthermore, if modelling other features besides the temporal constraints as controllable discrete variables, for instance, resource constraints or partial order alternations, the constraint model can help to solve scheduling or planning problems with flexible solutions that can make the execution robust.

The constraint model can only formulate the dynamic options with fixed allocations of uncertainty, which makes the solutions not optimal. Additionally, the current model is a baseline method. The run-time performance and memory use limit the scalability and usage of the method. In the future work, it can be enhanced by implementing other approaches, for instance, a conflict-directed search that starts with an infeasible solution performing best in terms of the objective function, finding and resolving conflicts and combining the found conflict resolutions until finding the optimal and feasible solution. A local method may also be able to formulate the dynamic allocation of uncertainty with dynamic options.

Chapter 6

Conclusion

In this thesis, we introduced the optimisation models of temporal problems with uncertainty and controllable options under constraints of varying levels of controllability. In the field of temporal problems with uncertainty, we focus on the optimisation models to answer how flexible, robust and controllable a problem is. Additionally, the optimisation model can show how much improvements can be achieved by making decision dynamically controllable, so that the dynamically controllable solutions can provide more flexibility to tackle the uncertain situations than fixed solutions which are strongly controllable.

We summarise the contributions of the thesis in Section 6.1. Section 6.2 presents the related work. Section 6.3 discusses the future work.

6.1 Summary of Contributions

We summarise the contribution of the thesis as follows.

- We introduce a disjunctive linear constraint model for STNU under dynamic controllability, to optimise the flexibility and robustness of temporal problems with uncertainty. The disjunctive linear constraint model can be encoded into MIP and non-linear programming, so that using existing solvers, such as GUROBI, CPLEX, IPOPT et al., can solve the models.

- We extend the definition and verification of dynamic controllability to temporal problems with uncertainty and controllable options, a CCTPU, which can solve the problem with an entirely dynamically controllable strategy in which both temporal scheduling and controllable options are decided dynamically. The dynamic decisions on the discrete options enable a fully flexible strategy in which every decision based on past observations.

- We try to introduce optimisation models of CCTPU under dynamic controllability with fixed or dynamic options, so that the optimisation model can answer how flexible or controllable a temporal problem with uncertainty and controllable options is in a dynamic strategy. Although the model introduced cannot describe the fully dynamic strategy as the checking algorithm introduced, regarding the assignment to discrete variables, the model can represent the dy-

namic decision based on a fixed division of the uncertainty in the past, which is still more flexible than the a fixed assignment.

- Robustness metrics are introduced based on the optimisation models. They can measure partial order schedules, partial order temporal plans and any temporal networks with uncertainty and discrete options. The metrics can answer how well the executions of those temporal networks can achieve in the environment with uncertainty.

- The optimisation models also allow us to evaluate the improvements from making the temporal networks strongly controllable to dynamically controllable with fixed assignment, and from dynamically controllable with fixed assignments to dynamic assignments.

6.2 Related Work

In addition to the techniques and methods that we used as the basis of the thesis, which have been discussed in the Background chapter, we present the related work in this section.

6.2.1 Related Temporal Reasoning Models

Among the variety of temporal reasoning models, the Conditional Simple Temporal Network with Uncertainty (CSTNU) is close to CCTPU. Both CSTNU and CCTPU consider temporal problems with uncertainty and conditions, but the CSTNU considers uncontrollable and observable conditions. Addition to that, we discuss two extensions of CSTNU in this section.

6.2.1.1 Conditional Simple Temporal Network with Uncertainty

Conditional Simple Temporal Network with Uncertainty (CSTNU) [Hunsberger et al., 2012] combines the Conditional Simple Temporal Network (CSTN) and Simple Temporal Network with Uncertainty (STNU). CSTN is a Conditional Temporal Problem (Definition 2.3) without disjunctive constraints.

Definition 6.1. A **CSTNU** is a tuple, $< V, E, L, OV, O, P, \ell >$, where:

- $< V, E, L, OV, O, P >$ is a CSTN which is a specified CTP with only binary constraints and without disjuncti ons,

- $(V, \lfloor E \rfloor, \ell)$ is an STNU, in which $\lfloor E \rfloor$ is the set of unlabelled constraints and ℓ is the set of contingent links,

- for each $(A, x, y, C) \in \ell$, $L(A) = L(C)$, and its labelled constraint is $(x \leq C - A \leq y, L(A))$.

The uncertain situations in the CSTNU is a combination of the uncertain conditions and the uncertain duration of the contingent links, which is called **drama** ($SC \times \Omega$), where SC is the same as the *execution scenario* of a CTP and $\omega \in \Omega$ denoted fixing duration of each contingent link which is the same as the *projection* of an STNU. The projection of CSTNU onto a drama (sc, ω) denoted by $drPrj(sc, \omega)$ is an STN. An **execution strategy** for CSTNU is a mapping from a drama to a complete set of assignments to controllable time points.

The controllability of CSTNU considers how well the execution strategy can deal with the drama. If an execution strategy is able to solve every drama of the CSTNU, it is dynamically controllable.

An algorithm for checking the dynamic controllability of a CSTNU is introduced by Combi et al. [2014]. It is based on the algorithm for checking dynamic controllability of STNU [Morris et al., 2001] which has a complexity of $O(N^5)$. The authors revised the reduction rules by considering the consistency of labels and adding label modification rules to the process to tackle the conditions of the CSTNU.

6.2.1.2 Conditional Disjunctive Temporal Networks with Uncertainty

Another temporal reasoning model that considers uncertainty and options is the Disjunctive Temporal Networks with Uncertainty (DTNU) [Venable and Yorke-Smith, 2005]. The DTNU extends STNU by considering disjunctive temporal constraints. Each requirement constraint is a disjunction of temporal constraints between any pairs of nodes. Each contingent link has a set of contingent intervals that do not have overlaps.

The uncertain situations and schedules of the DTNU are not very different from those of the STNU. A strongly controllable DTNU has a time assignment to every controllable node that ensures all constraints will be satisfied [Peintner et al., 2007]. The strong controllability of disjunctive temporal problems can be solved by SMT [Cimatti et al., 2015]. A dynamically controllable DTNU has a strategy that, given any uncertain situations, at least one disjunct on each constraint is satisfied [Venable et al., 2010; Cimatti et al., 2016b].

We can transfer every DTNU to a CCTPU by introducing a discrete variable for each temporal constraint that has disjunctions and adding labels accordingly. However, there is no apparent way to transform every CCTPU to DTNU since the constraints in the CCTPU can be transformed into a disjunction of conjunctions of temporal constraints but the DTNU is not able to represent the "conjunctions".

The Conditional Disjunctive Temporal Network with Uncertainty (CDTNU) is a combination of CSTNU and DTNU, which models uncontrollable conditions and disjunctive constraints at the same time [Cimatti et al., 2016a].

6.2.1.3 Conditional Simple Temporal Networks with Uncertainty and Decisions

Zavatteri [2017] introduced a temporal reasoning model called Conditional Simple Temporal Networks with Uncertainty and Decisions (CSTNUD) that consider uncon-

trollable and controllable conditions at the same time. The controllable conditions are named *decisions* in the CSTNUD.

Definition 6.2. A **CSTNUD** is a tuple, $< V, E, L, OV, DV, O, P, \ell, O >$, where:

- $< V, E, L, OV, O, P, \ell >$ is a CSTNU,
- DV is a set of decision time points such that $DV \cap OV = \emptyset$,
- $O : P \rightarrow OV \cup DV$ is a bijection associating a unique observation or decision time point to each proposition. If $O(p) \in OV$, p is observable, whereas if $O(p) \in DV$, p is decidable.

The proposition set $P = OP \cup DP$, where OP are the uncontrollable conditions that are observable, and DP are the controllable conditions.

The temporal reasoning model CCTPU, we discussed in the thesis, also considers decisions and uncertainty of temporal problems. However, the models of CCTPU and CSTNUD have some differences. The CCTPU model does not consider the uncontrollable conditions. Additionally, instead of giving a fixed decision time point to each controllable decisions, in CCTPU, we define a flexible mapping from the discrete variables to their decision time points in the strategy. When developing the dynamic controllability checking algorithm, we introduce some assumptions on the decision time points to achieve flexible and reasonable solutions.

6.2.2 Verification Approaches for Temporal Problems with Uncertainty

The checking algorithms and optimisation models in this thesis are based on the constraint programming technique – bound propagation. However, alternative representation like the timed game automata is also able to represent controllability of temporal problems.

Besides the temporal reasoning models and dynamic controllability representations, the frameworks verifying temporal models, such as CIRCA, UPPAAL et al., are discussed at the end of this section.

6.2.2.1 Representing Dynamic Controllability by Timed Game Automata

Cimatti et al. [2016a] used the Timed Game Automata (TGA) to represent the dynamic controllability of STNU, CSTNU, DTNU and CDTNU. Using TGA to represent the dynamic controllability is the first sound and complete approach for DTNU and CDTNU. Hunsberger and Posenato [2016] used TGA to represent the dynamic controllability of the CSTN. Zavatteri [2017] also used TGA to represent the dynamic controllability of the CSTNUD and implemented the model in UPPAAL-TIGA [Behrmann et al., 2007] which is an extension of the toolbox for modeling, simulation and verification of real-time systems – UPPAAL [Behrmann et al., 2006].

A Timed Automaton (TA) [Alur and Dill, 1990] is a tuple $< \sum, S, S_0, C, E >$, which adds a finite set of real-valued clocks C to a finite automaton $< \sum, S, S_0, E >$, where $\sum$ is an alphabet, S is the finite set of states, S_0 is the initial state and E are

the transitions. A transition represents the change from one state to another state on input. Each transition of a TA has a *guard* that describes the requirement to make the transition, and a set of clocks that will be reset when executes the transition.

A Timed Game Automaton (TGA) [Maler et al., 1995] divides the set of transitions into controllable and uncontrollable sets. Formulating the dynamic controllability of STNU by using TGA, Cimatti et al. [2016a] used two locations (same as the states in TA) to represent the states in which the agent or environment respectively execute its transitions, a goal location to model the agent achieves the goal, the uncontrollable transitions of the TA to model the executions of controllable nodes in the temporal networks and the controllable transitions to model the executions of contingent links. The clocks in the TGA of an STNU consist of a global one, a temporal one and one separate clock representing when each node is executed. Thus, the agent aims to find a counter-strategy to reach the goal location against the prevention from the environment.

Considering the uncontrollable conditions of the CSTNU, the TGA adds a clock for each proposition and a controllable transition for the location representing the environment that resets the clock, which represents that the observation shows true of the proposition. The same transitions for the location representing the agent model the decisions in CSTNUD.

The TGA is able to represent the sound and complete verification of dynamic controllability for temporal reasoning models since it models in a two-player game way, which simulates a real-time execution. Additionally, the verification tools for real-time systems (e.g. UPPAAL [Behrmann et al., 2006], Kronos [Yovine, 1997]) is able to implement the TGA formulation. Cassez et al. [2005] is the algorithm well used in TGA.

However, verifying dynamic controllability by using bound propagations may be more efficient. Zavatteri [2017] solved an example in a setting of two decisions, 4 contingent links and 11 nodes by implementing the TGA in UPPAAL-TIGA, which took about 1 minute in a virtual machine with Intel i7 2.8GHz CPU and 5G RAM. We test the same setting in a desktop with Intel i5 3.2GHz CPU and 4G RAM. Solving the same problem by using the application in Chapter 4 needs less than 1 second. Furthermore, solving the optimisation problem (we use the relaxing over-constrained problem) in the same setting takes 5 seconds. We cannot provide sufficient experiments to compare the two implementations since we do not have the implementation of the TGA and the details about the benchmark used in Zavatteri [2017].

6.2.3 Other Approaches for Temporal Problem Verification

Many real-time computing systems provide a reactive strategy or proactive-reactive approach to deal with the problems that have unexpected changes in the environment [Herroelen and Leus, 2005]. The proactive-reactive process is widely used since it enables the flexible adjustments under a proactive guidance.

Musliner et al. [1993] used a two-level architecture to build a real-time control system focusing on meeting hard deadlines – CIRCA. CIRCA has an AI subsystem

that can meet the task level goal and a real-time subsystem that can achieve the controllability. It sacrifices the completeness of the task-level calculation to guarantee the precision in the real-time system. Muscettola et al. [1998] developed a real-time system that focuses on tight deadlines, resource constraints and concurrent activities for the spacecraft domain that works over long periods of time. Shin and Ramanathan [1994] summarised a set of the real-time systems that use different techniques trading-off among efficiency, performance, completeness or precision. However, the dynamically controllable solutions for a temporal problem achieves the task-level completeness with the assumption that the uncertain durations are within certain bounds. It means that the techniques in dynamic controllability can be treated as the proactive process with assumptions of the reactive process. At the same time, it guarantees that there is a feasible solution when the execution satisfies the assumption. Using flexibility metrics related to dynamic controllability in a real-time system is an intelligent way to consider robustness. For example, for a scheduling problem, adding temporal slacks based on the dynamic controllability constraints is smarter than adding the slacks uniformly.

However, the assumptions that using intervals to describe the durations of the uncertain events sacrifice the accuracy. Thus the precise real-time systems like CIRCA is more useful in the applications that rely on a high precision than the methods describing durations in intervals, such as giving dynamically controllable solutions for STPU.

6.3 Future Work

In this thesis, we begin with an optimisation model of STPU which aims to answer how robust a temporal network with uncertainty is under the constraints of different levels of controllability. After that, we extend the dynamic controllability definition to temporal problems with uncertainty and controllable discrete variables and provide a checking algorithm. In the verification algorithm, we use assumptions to constrain the dynamic decision timepoints on when to assign the values to the discrete variables, so that the assignments can be assigned dynamically based on the observation of the past uncertainty. In the last technical chapter, we try to formulate the optimisation model for temporal problems with uncertainty and controllable discrete variables, in which constraints can represent different levels of controllability. However, the current constraints can only describe the fully strong controllability which makes both temporal schedules and variable assignments strongly controllable, the dynamic controllability in temporal schedules with fixed variable assignments. Representing a fully dynamic strategy in disjunctive linear constraint model is still an open question. We have tried to introduce a sub-optimal constraint model that can describe a dynamic strategy in which the temporal decisions and variable assignments are dynamically made, but the envelope for each option under a certain prehistory is fixed. Although the current constraint model cannot describe the fully dynamic strategy like the checking algorithm can do, the solutions of the sub-optimal

constraint model can provide the detail of some fully dynamic solutions that have better performance than constraint model of temporal dynamic controllability with fixed options.

In the future, we can try to improve the current work in the following directions.

Since the dynamic controllability verification algorithm is under a set of assumptions, relaxing those assumptions is one of the future works. Despite the assumptions that can be relaxed by remodelling the problem, providing more intelligent approaches that can describe the dynamic decision timepoints are useful extensions.

The constraint model in Chapter 5 is a baseline approach to solve the optimisation problem. Its efficiency and optimality limit the usage in real-world problems. Introducing a new optimisation method base on a local-search algorithm may be more efficient so that it can solve larger scale problems to enhance the usefulness of the model. One possible way is to build an algorithm based on the dynamic controllability verification algorithm in Chapter 4 since the dynamically controllable envelopes for each option after a certain prehistory is a disjunction of conjunctions of constraints. Optimising variables of bounds on the set of constraints can be solved by an SMT solver. Another possible method is to use the conflict-directed search (like CDRU), at the same time of iteratively finding and resolving conflicts, try to build an entirely dynamic strategy that can avoid the found conflicts by making options dynamically.

Extending the verification and optimisation of dynamic controllability to more general decision-making problems such as scheduling and planning is a critical future work. The controllable discrete variables can resolve the resource allocation conflicts that some activities cannot be executed at the same time because the sum of their resource demands exceeds the resource limitation. Different ways to add the precedence constraints that can resolve the conflict is the options for a controllable discrete variable. Using the CCTPU to represent a set of Partial Order Schedules to a scheduling problem enhances the flexibility contained in a solution. More robust or flexible schedules may be achieved by modeling the resource conflicts into a temporal network using CCTPU. The method in this thesis can verify if such a problem dynamically controllable or not and optimise the problem in different levels of controllability. However, encoding all resource constraints in a dynamically controllable strategy is a future work. Furthermore, planning problems are more complicated than the scheduling problems. Currently, one possible extension is to use controllable discrete variables to describe the alternations of plans solved by planners, then analysing robustness, controllability and other performances.

Last but not least, applying the dynamically controllable techniques to real autonomous systems is a goal of the topic. Before implementing the techniques to the real world, testing the strategies in simulation systems and robotic systems and improving its scalability and robustness are future work.

Bibliography

Allen, J. F., 1983. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26, 11 (Nov. 1983), 832–843. (cited on page 7)

Aloulou, M. A. and Portmann, M.-C., 2003. An efficient proactive reactive scheduling approach to hedge against shop floor disturbances. In *Proc. 1st Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA)*, 337–362. (cited on pages xvii, 21, 49, and 53)

Alur, R. and Dill, D., 1990. Automata for modeling real-time systems. In *Automata, Languages and Programming: 17th International Colloquium Warwick University, England, July 16–20, 1990 Proceedings*, 322–335. Springer Berlin Heidelberg, Berlin, Heidelberg. (cited on page 108)

Banerjee, D. and Haslum, P., 2011. Partial-order support-link scheduling. In *Proc. 21st International Conference on Automated Planning and Scheduling (ICAPS)*, 307–310. (cited on pages 49 and 51)

Bartusch, M.; Mohring, R. H.; and Radermacher, F. J., 1988. Scheduling project networks with resource constraints and time windows. *Ann. Oper. Res.*, 16, 1-4 (Jan. 1988), 201–240. (cited on page 20)

Beaumont, M.; Sattar, A.; Maher, M.; and Thornton, J., 2001. Solving Over-constrained Temporal Reasoning Problems. In *AI 2001: Advances in Artificial Intelligence*, 37–49. Springer Berlin Heidelberg, Berlin, Heidelberg. (cited on page 3)

Beaumont, M.; Thornton, J.; Sattar, A.; and Maher, M., 2004. Solving over-constrained temporal reasoning problems using local search. In *PRICAI 2004: Trends in Artificial Intelligence. 8th Pacific Rim International Conference on Artificial Intelligence. Proceedings, 9-13 Aug. 2004*, 134. Springer Berlin Heidelberg, Berlin, Heidelberg. (cited on page 3)

Behrmann, G.; Cougnard, A.; David, A.; Fleury, E.; Larsen, K. G.; and Lime, D., 2007. Uppaal-tiga: Time for playing games! In *Computer Aided Verification: 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007. Proceedings*, 121–125. Springer Berlin Heidelberg, Berlin, Heidelberg. (cited on page 108)

Behrmann, G.; David, A.; Larsen, K. G.; Hakansson, J.; Petterson, P.; Yi, W.; and Hendriks, M., 2006. Uppaal 4.0. In *Quantitative Evaluation of Systems, 2006. QEST 2006. Third International Conference on*, 125–126. IEEE. (cited on pages 108 and 109)

Boerkoel, J. C. and Durfee, E. H., 2013. Decoupling the multiagent disjunctive temporal problem. In *In Proceedings of the Twenty-Seventh Conference on Artificial Intelligence (AAAI-13)*. (cited on page 22)

Cassez, F.; David, A.; Fleury, E.; Larsen, K. G.; and Lime, D., 2005. Efficient On-the-Fly Algorithms for the Analysis of Timed Games. In *CONCUR 2005 – Concurrency Theory*, 66–80. Springer Berlin Heidelberg, Berlin, Heidelberg. (cited on page 109)

Cesta, A.; Oddi, A.; and Smith, S. F., 1998. Profile-based algorithms to solve multiple capacitated metric scheduling problems. In *Proc. 4th International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*, 214–223. (cited on page 22)

Cimatti, A.; Hunsberger, L.; Micheli, A.; Posenato, R.; and Roveri, M., 2016a. Dynamic controllability via timed game automata. *Acta Informatica*, 53, 6 (Oct 2016), 681–722. (cited on pages 107, 108, and 109)

Cimatti, A.; Micheli, A.; and Roveri, M., 2015. Solving strong controllability of temporal problems with uncertainty using smt. *Constraints*, 20, 1 (2015), 1–29. (cited on page 107)

Cimatti, A.; Micheli, A.; and Roveri, M., 2016b. Dynamic controllability of disjunctive temporal networks: Validation and synthesis of executable strategies. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16 (Phoenix, Arizona, 2016), 3116–3122. AAAI Press. (cited on page 107)

Combi, C.; Hunsberger, L.; and Posenato, R., 2014. An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty - revisited. In *Proc. 5th International Conference on Agents and Artificial Intelligence (ICAART)*, 314–331. (cited on pages 57 and 107)

Conrad, P. R. and Williams, B. C., 2011. Drake: An efficient executive for temporal plans with choice. *J. Artif. Int. Res.*, 42, 1 (Sep. 2011), 607–659. (cited on page 61)

Cui, J. and Haslum, P., 2017. Dynamic controllability of controllable conditional temporal problems with uncertainty. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS)*, 61–69. (cited on pages 6 and 57)

Cui, J.; Yu, P.; Fang, C.; Haslum, P.; and Williams, B., 2015. Optimising bounds in simple temporal networks with uncertainty under dynamic controllability constraints. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, 52–60. (cited on pages 5 and 26)

Dechter, R.; Meiri, I.; and Pearl, J., 1991. Temporal constraint networks. *Artificial Intelligence*, 49 (1991), 61–95. (cited on pages 2, 7, 8, and 17)

Even, C.; Pillac, V.; and Hentenryck, P. V., 2014. Nicta evacuation planner: Actionable evacuation plans with contraflows. In *Proceedings of the Twenty-first European*

Conference on Artificial Intelligence (ECAI), ECAI'14 (Prague, Czech Republic, 2014), 1143–1148. IOS Press, Amsterdam, The Netherlands, The Netherlands. (cited on page 8)

Fang, C.; Yu, P.; and Williams, B. C., 2014. Chance-constrained probabilistic simple temporal problems. In *Proc. 28th AAAI Conference on Artificial Intelligence*, 2264–2270. (cited on pages 9, 11, 25, 46, 52, 53, and 54)

Gill, P. E.; Murray, W.; and Saunders, M. A., 2002. SNOPT: an SQP algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, 12, 4 (2002), 979–1006. (cited on page 48)

Goren, S. and Sabuncuoglu, I., 2008. Robustness and stability measures for scheduling: single-machine environment. *IIE Transactions*, 40, 1 (2008), 66–83. (cited on page 3)

Herroelen, W. and Leus, R., 2005. Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, 165, 2 (2005), 289 – 306. Project Management and Scheduling. (cited on page 109)

Hunsberger, L., 2002. Algorithms for a temporal decoupling problem in multi-agent planning. In *In Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI)*, 468–475. (cited on page 22)

Hunsberger, L., 2009. Fixing the semantics for dynamic controllability and providing a more practical characterization of dynamic execution strategies. In *Proc. 16th International Symposium on Temporal Representation and Reasoning (TIME)*, 155–162. (cited on pages 9, 15, 59, 63, and 85)

Hunsberger, L., 2013. Magic loops in simple temporal networks with uncertainty. In *Proceedings of the Fifth International Conference on Agents and Artificial Intelligence (ICAART)*, 332–350. (cited on pages 16, 19, and 70)

Hunsberger, L. and Posenato, R., 2016. A New Approach to Checking the Dynamic Consistency of Conditional Simple Temporal Networks. In *Principles and Practice of Constraint Programming*, 268–286. Springer International Publishing. (cited on page 108)

Hunsberger, L.; Posenato, R.; and Combi, C., 2012. The dynamic controllability of conditional STNs with uncertainty. In *Proc. Planning and Plan Execution for Real-World Systems: Principles and Practices (PlanEx) Workshop*, 2–4. (cited on pages 4, 9, 57, and 106)

Jorge Leon, V.; David Wu, S.; and Storer, R. H., 1994. Robustness measures and robust scheduling for job shops. *IIE Transactions*, 26, 5 (1994), 32–43. (cited on page 3)

Kolisch, R. and Padman, R., 2001. An integrated survey of project scheduling. *OMEGA International Journal of Management Science*, 29, 3 (2001), 249–272. (cited on page 51)

Maler, O.; Pnueli, A.; and Sifakis, J., 1995. On the synthesis of discrete controllers for timed systems. In *STACS 95: 12th Annual Symposium on Theoretical Aspects of Computer Science Munich, Germany, March 2–4, 1995 Proceedings*, 229–242. Springer Berlin Heidelberg, Berlin, Heidelberg. (cited on page 109)

Morris, P., 2006. A structural characterization of temporal dynamic controllability. In *Proc. 12th International Conference on Principles and Practice of Constraint Programming (CP)*, 375–389. (cited on pages 19, 70, and 83)

Morris, P., 2014. Dynamic controllability and dispatchability relationships. In *Proc. 11th Integration of AI and OR Techniques in Constraint Programming (CPAIOR)*, 464–479. (cited on pages 19, 70, and 83)

Morris, P. and Muscettola, N., 2000. Execution of temporal plans with uncertainty. In *In Proceedings of the 17th National Conference on Artificial Intelligence (AAAI)*, 491–496. (cited on pages 16 and 85)

Morris, P. and Muscettola, N., 2005. Temporal dynamic controllability revisited. In *In Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-2005*, 1193–1198. AAAI Press / The MIT Press. (cited on pages 17, 18, and 19)

Morris, P.; Muscettola, N.; and Vidal, T., 2001. Dynamic control of plans with temporal uncertainty. In *Proc. 17th International Conference on Artificial Intelligence (IJCAI)*, 494–499. (cited on pages 9, 16, 25, 27, 29, 55, 59, 63, and 107)

Mountakis, S.; Klos, T.; and Witteveen, C., 2015. Temporal flexibility revisited: Maximizing flexibility by computing bipartite matchings. In *International Conference on Automated Planning and Scheduling*. (cited on page 24)

Muise, C., 2014. *Exploiting Relevance to Improve Robustness and Flexibility in Plan Generation and Execution*. Ph.D. thesis, University of Toronto. (cited on page 1)

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C., 1998. Remote agent: To boldly go where no ai system has gone before. *Artif. Intell.*, 103, 1-2 (Aug. 1998), 5–47. (cited on page 110)

Musliner, D. J.; Durfee, E. H.; and Shin, K. G., 1993. Circa: a cooperative intelligent real-time control architecture. *IEEE Transactions on Systems, Man, and Cybernetics*, 23, 6 (Nov 1993), 1561–1574. (cited on page 109)

Nilsson, M.; Kvarnström, J.; and Doherty, P., 2013. Incremental dynamic controllability revisited. In *Proceedings of the 13th International Conference on International Conference on Automated Planning and Scheduling (ICAPS)*, 337–341. (cited on page 70)

Nilsson, M.; KvarnstrÃűm, J.; and Doherty, P., 2014. Incremental dynamic controllability in cubic worst-case time. In *Proc. 21st International Symposium on Temporal Representation and Reasoning (TIME)*, 17–26. (cited on pages 16 and 19)

Peintner, B.; Venable, K. B.; and Yorke-Smith, N., 2007. Strong controllability of disjunctive temporal problems with uncertainty. In *Principles and Practice of Constraint Programming (CP) 2007*, 856–863. Springer Berlin Heidelberg, Berlin, Heidelberg. (cited on page 107)

Policella, N.; Cesta, A.; Oddi, A.; and Smith, S., 2009. Solve-and-robustify. *Journal of Scheduling*, 12 (2009), 299–314. (cited on page 49)

Policella, N.; Cesta, A.; Oddi, A.; and Smith, S. F., 2007. From precedence constraint posting to partial order schedules: A csp approach to robust scheduling. *AI Communications, Special Issue on Constraint Programming for Planning and Scheduling*, (May 2007), 163–180. (cited on page 1)

Policella, N.; Smith, S.; Cesta, A.; and Oddi, A., 2004. Generating robust schedules through temporal flexibility. In *Proc. 14th International Conference on Automated Planning & Scheduling (ICAPS)*, 209–218. (cited on pages 20, 21, and 23)

Schwalb, E. and Vila, L., 1998. Temporal constraints: A survey. *Constraints*, 3, 2-3 (1998), 129–149. (cited on page 7)

Shah, J. A.; Stedl, J.; Williams, B. C.; and Robertson, P., 2007. A fast incremental algorithm for maintaining dispatchability of partially controllable plans. In *Proceedings of the Seventeenth International Conference on International Conference on Automated Planning and Scheduling (ICAPS)*, 296–303. (cited on pages 16 and 70)

Shin, K. G. and Ramanathan, P., 1994. Real-time computing: a new discipline of computer science and engineering. *Proceedings of the IEEE*, 82, 1 (Jan 1994), 6–24. (cited on page 110)

Tsamardinos, I., 2002. A probabilistic approach to robust execution of temporal plans with uncertainty. *Methods and Applications of Artificial Intelligence*, (April 2002), 97–108. (cited on pages 9, 11, and 52)

Tsamardinos, I.; Vidal, T.; and Pollack, M. E., 2003. CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints*, 8, 4 (2003), 365–388. (cited on pages 4, 9, and 11)

Venable, K. B.; Volpato, M.; Peintner, B.; and Yorke-Smith, N., 2010. Weak and dynamic controllability of temporal problems with disjunctions and uncertainty. In *Proc. of the Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS-20910) in ICAPS-2010*, 50–59. (cited on page 107)

Venable, K. B. and Yorke-Smith, N., 2005. Disjunctive temporal problems with uncertainty. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05 (Edinburgh, Scotland, 2005), 1721–1722. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. (cited on page 107)

Vidal, T. and Fargier, H., 1999. Handling contingency in temporal constraint networks: From consistency to controllabilities. *Journal of Experimental and Theoretical AI*, 11, 1 (1999), 23–45. (cited on pages 2, 3, 9, 10, 14, 16, 25, 26, 47, 59, 60, and 63)

Vidal, T. and Ghallab, M., 1996. Dealing with uncertain durations in temporal constraint networks dedicated to planning. In *Proc. 12th European Conference on Artificial Intelligence (ECAI)*, 48–52. (cited on pages 19, 25, 26, and 46)

Vilain, M.; Kautz, H.; and Beek, P., 1986. Constraint propagation algorithms for temporal reasoning. In *Readings in Qualitative Reasoning about Physical Systems*, 377–382. Morgan Kaufmann. (cited on page 7)

Wah, B. W. and Xin, D., 2004. Optimization of bounds in temporal flexible planning with dynamic controllability. In *Proc. 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 40–48. (cited on pages 25, 32, 33, 34, 36, 40, 43, 51, and 52)

Wah, B. W. and Xin, D., 2007. Optimization of bounds in temporal flexible planning with dynamic controllability. *International Journal on Artificial Intelligence Tools*, 16, 1 (2007), 17–44. (cited on page 32)

Wang, A. J. and Williams, B. C., 2015. Chance-constrained scheduling via conflict-directed risk allocation. In *Proc. 29th AAAI Conference on Artificial Intelligence (AAAI)*, 3620–3627. (cited on page 11)

Weld, D., 1994. Introduction to least commitment planning. *AI Magazine*, 15, 4 (1 1994), 27–61. (cited on page 1)

Williams, B. C. and Ragno, R. J., 2002. Conflict-directed A* adn its role in model-based embedded systems. *Journal of Discrete Applied Mathematics*, 155, 12 (2002), 1562–1595. (cited on page 44)

Wilson, M.; Klos, T.; Witteveen, C.; and Huisman, B., 2014. Flexibility and decoupling in simple temporal networks. *Artificial Intelligence*, 214 (2014), 26–44. (cited on pages xvii, 23, 24, and 53)

Wu, S. D.; Byeon, E.-S.; and Storer, R. H., 1999. A graph-theoretic decomposition of the job shop scheduling problem to achieve scheduling robustness. *Operations Research*, 47, 1 (1999), 113–124. (cited on page 3)

Yovine, S., 1997. Kronos: A verification tool for real-time systems. (kronos user's manual release 2.2). *International Journal on Software Tools for Technology Transfer*, 1 (1997), 123–133. (cited on page 109)

Yu, P., 2016. BCDR Test Generator. https://github.com/yu-peng/BCDRTestGenerator. (cited on page 86)

Yu, P.; Fang, C.; and Williams, B. C., 2014. Resolving uncontrollable conditional temporal problems using continuous relaxations. In *Proc. 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 341–349. (cited on pages 4, 9, 14, 25, 33, 44, 48, 55, 57, 59, 60, 67, 68, 69, 86, and 95)

Yu, P. and Williams, B., 2013. Continuously relaxing over-constrained conditional temporal problems through generalized conflict learning and resolution. In *Proc. 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, 2429–2436. (cited on pages 9, 13, 44, and 86)

Yu, P.; Williams, B.; Fang, C.; Cui, J.; and Haslum, P., 2017. Resolving over-constrained temporal problems with uncertainty through conflict-directed relaxation. *Journal of Artificial Intelligence Research*, 60 (2017), 425–490. (cited on page 5)

Zavatteri, M., 2017. Conditional Simple Temporal Networks with Uncertainty and Decisions. In *24th International Symposium on Temporal Representation and Reasoning (TIME 2017)*, vol. 90 of *Leibniz International Proceedings in Informatics (LIPIcs)*, 23:1–23:17. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany. (cited on pages 107, 108, and 109)