Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2017-001 October 14, 2016

Collaborative Diagnosis of Over-Subscribed Temporal Plans

Peng Yu

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Collaborative Diagnosis of Over-Subscribed Temporal Plans

by

Peng Yu

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2017

Author .....
Department of Aeronautics and Astronautics
October 15, 2016

Certified by .....
Brian C. Williams
Professor
Thesis Supervisor

Certified by .....
Leslie P. Kaelbling
Professor
Thesis Committee Member

Certified by .....
Randall Davis
Professor
Thesis Committee Member

Accepted by .....
Paulo C. Lozano
Chair, Graduate Program Committee

Collaborative Diagnosis of Over-Subscribed Temporal Plans

by

Peng Yu

Submitted to the Department of Aeronautics and Astronautics
on October 15, 2016, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Over-subscription, that is, being assigned too many tasks or requirements that are too demanding, is commonly encountered in temporal planning problems. As human beings, we often want to do more than we can, ask for things that may not be available, while underestimating how long it takes to perform each task. It is often difficult for us to detect the causes of failure in such situations and then find resolutions that are effective. We can greatly benefit from tools that assist us by looking out for these plan failures, by identifying their root causes, and by proposing preferred resolutions to these failures that lead to feasible plans.

In recent literature, several approaches have been developed to resolve such over-subscribed problems, which are often framed as over-constrained scheduling, configuration design or optimal planning problems. Most of them take an all-or-nothing approach, in which over-subscription is resolved through suspending constraints or dropping goals. While helpful, in real-world scenarios, we often want to preserve our plan goals as much possible. As human beings, we know that slightly weakening the requirements of a travel plan, or replacing one of its destinations with an alternative one is often sufficient to resolve an over-subscription problem, no matter if the requirement being weakened is the duration of a deep-sea survey being planned for, or the restaurant cuisine for a dinner date.

The goal of this thesis is to develop domain independent relaxation algorithms that perform this type of slight weakening of constraints, which we will formalize as continuous relaxation, and to embody them in a computational aid, Uhura, that performs tasks akin to an experienced travel agent or ocean scientists. In over-subscribed situations, Uhura helps us diagnose the causes of failure, suggests alternative plans, and collaborates with us in order to resolve conflicting requirements in the most preferred way. Most importantly, the algorithms underlying Uhura supports the weakening, instead of suspending, of constraints and variable domains in a temporally flexible plan.

The contribution of this thesis is two-fold. First, we developed an algorithmic framework, called Best-first Conflict-Directed Relaxation (BCDR), for performing plan relaxation. Second, we use the BCDR framework to perform relaxation for sev-

eral different families of plan representations involving different types of constraints. These include temporal constraints, chance constraints and variable domain constraints, and we incorporate several specialized conflict detection and resolution algorithms in support of the continuous weakening of them. The key idea behind BCDR's approach to continuous relaxation is to generalize the concepts of discrete conflicts and relaxations, first introduced by the model-based diagnosis community, to hybrid conflicts and relaxations, which denote minimal inconsistencies and minimal relaxations to both discrete and continuous relaxable constraints.

In addition, we present the design and implementation of Uhura, the integrated plan advisory system that incorporates BCDR for resolving over-subscribed temporal plans. Uhura can efficiently produce a relaxed plan for the user to support multiple, interrelated constraints and activities. We have applied Uhura to different types of plans to illustrate the practical generality of our approach, which includes deep-sea exploration, job-shop scheduling and transit system management. Results from the computational experiments we performed also show that BCDR is one to two orders of magnitude faster than existing algorithms that build on state-of-the-art numerical solvers, making it an effective approach for many large-scale plans in the aforementioned domains.

Thesis Supervisor: Brian C. Williams
Title: Professor

Thesis Committee Member: Leslie P. Kaelbling
Title: Professor

Thesis Committee Member: Randall Davis
Title: Professor

Acknowledgments

I would like to express my gratitude to all who have supported me in the completion of my research projects and this thesis.

First of all, I would like to express my gratitude towards my supervisor, Professor Brian C. Williams, for his endless support and encouragement in the past six years. I came to MIT without any background in model-based autonomy and artificial intelligence. He introduced me to the field, pointing in the right direction for my research and has provided invaluable feedback through numerous meetings and conversations, without which my thesis would not have been possible. I also want to thank my committee members, Professor Leslie Kaelbling and Professor Randall Davis, for their inputs and guidance throughout the thesis development and writing process.

I would like to thank every member in the MERS group for their insightful comments and discussions throughout my thesis writing, and for their support in my research during the past six years. Specifically, I thank David Wang and Eric Timmons for their tireless help and answers to all my problems and questions, saving me from many frustrating situations, Erez Karpas, Christian Muise and Andreas Hofmann for mentoring me during the critical moments of my graduate research, helping me overcome many challenges in my projects and thesis. Simon, Steve, Szymon, Jonathan, Enrique, Tiago, James, Pedro, Dan, Andrew, Shannon, Ameya, Hiro, Spencer, Sean, Wesley, Larry, Bobby, Nikhil, Yuening, Jingkai, Cyrus, Matt and Sang, I am grateful to have the opportunity to work with all of you. You make MERS a great team, and my PhD adventure exciting and enjoyable.

I would also like to thank all of my collaborators at Boeing Research & Technology, Woods Hole Oceanographic Institute Deep Submergence Lab, Nuance Natural Language and Artificial Intelligence Laboratory and National ICT of Australia, for their guidance and assistance in my research projects. A big thanks goes to Ron Provine and Scott Smith at Boeing, for setting me down this path, and providing a rich set of use cases for the plan relaxation research; and to Jiaying Shen and Peter Z. Yeh at Nuance, for mentoring me during my internship and providing valuable

insights on realizing the domain relaxation capability. I would also like to thank Rich Camilli, for giving me the opportunity to expand the application of my research to deep-sea exploration, and sharing his experience and insights with me. In addition, I would to thank Jing Cui and Patrik Haslum at NICTA, for their careful evaluations of my research and detailed feedbacks, which are key for the correctness and robustness of my thesis. Everything presented in this thesis will not be possible without all my collaborators' generous help.

Most importantly, I'd like to thank my parents Jie Cui and Zhihe Yu for their endless support and love throughout my life. Even though I am ten thousand kilometers away from home, you always encourage me to pursue my goals, whatever and wherever they may be. I also thank my wife, Zhuo Zhang, for her love and guidance that helps me to overcome the difficulties in my life. Without their help, there is no way for me to come to MIT for graduate study, achieving each milestone during the journey and finally completing this thesis.

Finally, I would like to thank my sponsor, the Boeing Company, for their generous support in the past years that makes this thesis possible. My research was supported under Boeing Company grant MIT-BA-GTA-1. Additional support was provided by the DARPA EdgeCT program.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

From an evening outing to a summer vacation, we frequently plan for travels of different length and complexity. Unfortunately, we are not good at estimating times, compensating for uncertainty and coordinating with other people. The problem becomes even more challenging when we are under time pressure. These situations can lead to anywhere from being late for a dinner, to missing a flight. Similar situations are often encountered in the operation of unmanned robotic systems, such as Autonomous Underwater Vehicles. From traversal times to weather conditions, uncertainty exists in every deep-sea expedition mission. The imperfect modeling and unbounded uncertainty in the environment, as well as the underwater vehicles and the crew performance, make it impossible to find a mission plan that offers a 100% guarantee of success. Therefore, correct handling of uncertainties and management of risk are essential requirements for the ocean scientist who manages expedition plans. When the situations become over-subscribed, the scientists have to quickly make trade-offs between scientific goals, mission requirements and risk to restore the feasibility of the mission. It would be of great help if there is an intelligent plan assistant that can keep us informed about such issues, and provide advice on which goals and requirements should be modified, such that a robust plan, no matter if it is for us humans or our robotic systems, can be generated.

Prior work on this issue starts with a scheduling model, which encodes such scenarios using over-constrained temporal problems. A temporal problem is over-

constrained if no execution strategy can be found that meets all constraints (Dechter, Meiri, & Pearl, 1991; Vidal & Fargier, 1999). To solve an over-constrained temporal problem, one has to identify its conflicting constraints and weaken some of them, such that all conflicts are resolved and a feasible execution strategy, either a static schedule or a dynamic policy, can be generated. In literature, several methods have been developed to solve such problems. (Beaumont, Sattar, Maher, & Thornton, 2001; Beaumont, Thornton, Sattar, & Maher, 2004) took a partial constraint satisfaction approach (Freuder & Wallace, 1992) to find subsets of satisfiable constraints for over-constrained Simple Temporal Problems (STPs). Later, disjunctive constraints and optimality were added in the context of over-constrained Disjunctive Temporal Problems with Preferences (DTPPs) (Moffitt & Pollack, 2005a, 2005b; Peintner, Moffitt, & Pollack, 2005). In a DTPP, the disjuncts of every constraint are assigned a preference function that maps the temporal constraint to a cost value. The optimal partial solution is obtained by enumerating consistent subproblems using Branch & Bound, as well as other optimization techniques introduced in (Khatib, Morris, Morris, & Rossi, 2001). Most of the prior work has focused on restoring consistency through complete suspension of constraints, however, in real-world scenarios, the user often wants to preserve as much of the schedule as possible.

One approach to address this issue is presented in (Rossi, Sperduti, Venable, Khatib, Morris, & Morris, 2002), which presents the formulation of Simple Temporal Problems with Preferences (STPPs). To allow the weakening for an over-constrained temporal problem, it introduces soft temporal constraints, which contains a disjunctive set of predefined temporal bounds. These bounds, associated with preference functions defined over the time assigned to each event, provides more alternatives for the scheduling algorithm to meet the feasibility requirements. (Khatib et al., 2001) demonstrates that finding the optimal solution to a STPP with semi-convex preferences is tractable. Later, (Rossi, Venable, & Yorke-Smith, 2006) introduces a generalization of the STPP formulation to include uncertain durations, and a suite of algorithms for finding the optimal solutions under strong, weak and dynamic controllability assumption. Beyond scheduling problems, researchers have also used a

much richer activity and constraint model for encoding the over-subscribed planning problems. For example, (Domshlak & Mirkis, 2015) presents several approximation techniques for deterministic oversubscription planning (OSP). In literature, optimal planning has been the primary approach for OSPs, in which the objective is reformulated as finding a plan to achieve a subset of the goals with higher rewards.

In addition to weakening the relations of constraints or dropping goals, there is also work on resolving over-subscription by introducing more options into the problem. In (Thompson, Goker, & Langley, 2004), a conversational recommendation system for point-of-interest selections is presented. It integrates a personalized preference model that updates through interactions with the users. Moreover, the approach presented in this paper addresses over-subscription along a different dimension: if the users ask for too much and no candidate place can meet their requirements, the presented system will propose domain constraints to drop in order to allow more candidates to be considered, effectively resolving the over-subscription. However, similar to prior works on over-constrained temporal problems, this is also an all-or-nothing approach, in which domain constraints are completely suspended if any of them are in conflict. While able to restore the feasibility of over-constrained problems, these suspensions are often not necessary. As human beings, we know that slightly weakening the constraints with an alternative one is often sufficient to resolve the issues.

This is the motivation for us, and the issue we address in this thesis: we would like to develop an autonomous system that behaves more like an experienced travel agent or expedition scientist. In over-subscribed situations, it will help humans diagnose the causes of failure, suggest alternative plans, and collaborate with us in order to resolve conflicting requirements in the most preferred way. More importantly, its reasoning algorithm supports the weakening, instead of suspending, of constraints such that the original plans can be preserved to the maximal extent.

Thesis Statement

Resolving over-subscribed temporal plans using a variety of efficient continuous relaxation techniques leads to greater flexibility in plan adaptation. Compared to discrete

relaxations, which suspend constraints completely, **continuous** relaxations for the temporal and domain constraints are often more preferred since they weaken the original requirements to the minimal extent. The key to efficient generation of continuous relaxations is to pinpoint the set of conflicting constraints, which denote minimal inconsistencies in the plan and minimal relaxations to both discrete and continuous relaxable constraints. The development of such a continuous relaxation capability pose four separate sub-problems:

1. The problem of detecting the exact cause of failure in over-subscribed temporal plans, and enumerate their relaxations in best-first order.

2. The problem of computing preferred continuous relaxations, instead of suspensions, for temporal bounds in over-subscribed temporal plans, based on a user preference model.

3. The problem of generating a robust and risk-bounded relaxations for plans under temporal uncertainty.

4. The problem of computing preferred relaxations, instead of complete removal, for domain constraints, based on a user preference model.

In this thesis, we present four main contributions to solve each of the problems. First, we present a novel framework for detecting and resolving conflicts in over-subscribed temporal plans, which builds upon prior work on conflict-directed diagnosis and is capable of enumerating discrete relaxations in best-first order. Second, we introduce the extension for computing continuous relaxations for temporal bounds in conflicts. The key of continuous relaxation is to generalize the discrete conflicts and relaxations, to hybrid conflicts and relaxations, which denote minimal inconsistencies and minimal relaxations to both discrete and continuous relaxable constraints. Third, we develop a set of algorithms for detecting conflicts that involve uncertain durations, and computing risk-bounded temporal relaxations. Fourth, we present the extension to the relaxation framework for computing domain relaxations, which

resolves conflicts by allowing more options to be considered in the destinations' domains. Domain relaxation extends the continuous weakening concept from temporal constraints to domain constraints, which explores candidates along a different dimension in situations where we cannot compromise on time. These contributions are summarized in Sections 1.1 through 1.4.

1.1 A Conflict-directed Relaxation Framework for Over-subscribed Temporal Plans

The first contribution of the thesis is a novel framework, Best-first Conflict-Directed Relaxation (BCDR), for detecting and resolving conflicts in over-subscribed temporal plans. Building upon prior work on diagnosis (de Kleer & Williams, 1987; Williams & Ragno, 2002) and over-constrained CSPs (Bailey & Stuckey, 2005; Moffitt & Pollack, 2005b), BCDR is capable of handling temporal plans with discrete and continuous variables and constraints. Instead of likely failure modes, it detects the causes of failure in the plans and supports the enumeration of preferred conflict resolutions in best-first order. Our goal is to develop a system that supports the resolution for more complex travel planning, deep-sea exploration and robotic manufacturing scenarios that may involve multiple activities and agents. The system will be applicable to a broad range of interesting use cases, and can answer user requests like where to meet, when to leave, how to get to the places, and how long to stay. In over-subscribed situations where some of the requirements cannot be met, we would like the system to propose preferred resolutions using alternatives for both destinations and timing, and preserve as much flexibility as possible for the users.

We first presented the framework in (Yu, Shen, Yeh, & Williams, 2016b), which takes plans encoded using the Temporal Plan Network (TPN) formalism as input, and produce a variety of partial relaxations and temporally feasible plans as output. This model is more general than the temporal problem formulations used in many prior works on relaxation, since it provides support for choices over alternative plans

and multi-agent coordination.

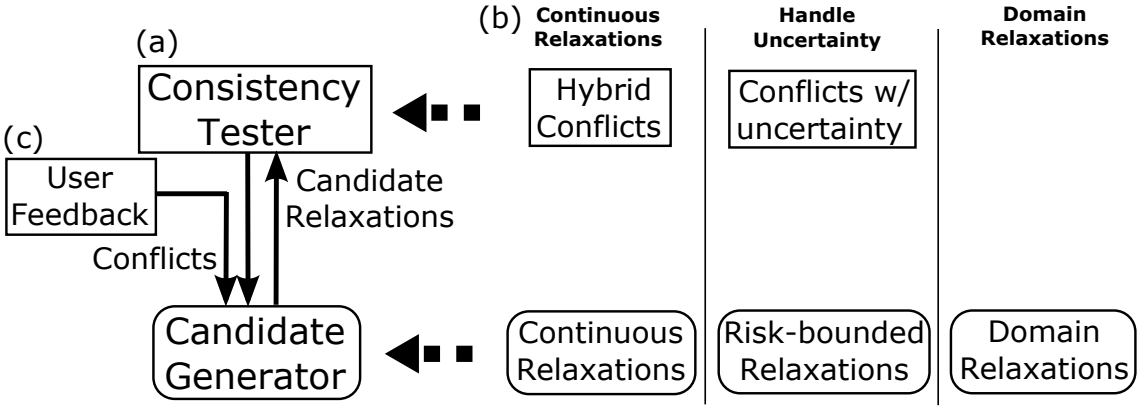


Figure 1-1: The generate and test architecture and extensions

The conflict-directed diagnosis algorithms we are building upon, General Diagnosis Engine and Conflict-directed A*, take a generate-and-test approach for finding consistent diagnosis for robotic systems. As shown in Figure 1-1a, there are two major components, Candidate Generator and Consistency Tester, in the framework. BCDR iterates between the two components until an agreement, which is a temporally feasible plan, is reached. Given a candidate solution, the consistency tester evaluates if all constraints are satisfied. If not, it produces a conflicting set of assignments and constraints as the explanation. The generator then incorporates the conflict, produces a new candidate solution that resolves it, and passes it over to the tester to evaluate its temporal feasibility again.

Our relaxation framework preserves this conflict-directed approach for its extensibility, transparency, and efficiency. First, the simple generate-test structure makes it very easy to incorporate new plan features and relaxation techniques. Second, the conflicts used to guide the search and candidate generation are also essential components in the solution presented to the users. They allow BCDR to explain not only the relaxations, but also the rationale behind them. The transparency enables more effective communication with the users, which will help them make more informed decisions. Third, conflict-directed search has been proven to be an efficient approach in hardware diagnosis, and in this thesis we also demonstrate its effectiveness in resolving large-scale over-subscribed temporal plans through experiments.

The approach taken by our framework can also be viewed as a specialization of Logic-based Benders Decomposition (Hooker & Ottosson, 2003), in which the master problem determines the activation of temporal constraints and the degree of their relaxations, while the subproblem checks if an execution policy exists given all activated constraints. The basic setup of the framework is only capable of handling *discrete relaxations*, which resolves any conflicts by suspending one or more of the constraints involved. For example, given that 30 minutes is not enough time for a person to grab lunch and have a hair cut, the discrete relaxations available are to either give up lunch or hair cut completely. While this not-so-preferred behavior is similar to prior works on over-subscription, it serves as the bases for the other three contributions of this thesis. Hence we start our presentation with this framework, then fold in the three other extensions into the tester and generator as plugins to support constraint weakening and risk-bounded relaxations. As shown in Figure 1-1b, to compute continuous relaxation, we generalized both the tester (for returning hybrid conflicts) and the generator (for computing continuous temporal relaxations). For problems that are subject to temporal uncertainty, the two components are further extended to extract conflicts involving uncertain durations, and for computing risk-bounded relaxations. Finally, domain relaxation is supported by extending the generator with the capability for weakening the constraints for variable domains.

Finally, note that the conflict-directed relaxation framework also supports the incorporation of user feedback. As the user models we work with are often imperfect, it is likely that the relaxations BCDR proposes to the users are not preferred because the users did not specify that one or more constraints should not have been relaxed at all, or weakened to a certain extent. Supporting this feature is in fact a very natural extension for the conflict-directed relaxation framework. As shown in Figure 1-1c, every user feedback is encoded as an additional conflict and pushed into the candidate generator, such that all future candidate solutions will respect the newly added feedback. In addition, these conflicts can be preserved and carried over from one problem to another, so that future iterations will try to avoid running into the same situation.

1.2 Continuous Relaxation for Temporal Constraints

The second contribution of this thesis is a novel method for computing continuous relaxations for conflicting temporal constraints. We previously introduced it in (Yu & Williams, 2013) as the Best-first Conflict-Directed Relaxation algorithm for scheduling problems. The continuous relaxation approach efficiently resolves over-subscribed temporal plans with controllable temporal variables. It reformulates an over-subscribed temporal plan by identifying its continuously relaxable temporal bounds, which can be partially relaxed to restore consistency. The key idea behind continuous relaxation is to generalize the discrete conflicts and relaxations in BCDR, to **hybrid conflicts** and relaxations, which denote minimal inconsistencies and minimal relaxations to both discrete and continuous relaxable constraints. Through learning hybrid conflicts, BCDR is able to use their resolutions for generating continuous relaxations that weaken the temporal bounds to the minimal extent, as well as guiding the search away from infeasible regions. The continuous relaxation approach is implemented as extensions to the consistency tester and candidate generator in BCDR.

For example, imagine that a graduate student, Simon, is planning for an evening outing trip with his friends. Simon is leaving his office at 6pm, and would like to have dinner with a friend at a Chinese restaurant then watch a comedy movie. He needs to be home before 9:30pm. We use a TPN to model Simon's travel plan and determine the best strategy for him that includes: which movie to watch, which restaurant to dine at, how much time to spend at each location and whether to delay his arrival at home. The TPN that encodes his travel problem is shown in (Figure 1-2), which contains the two activities for dinner and movie, the options available for them, and the temporal constraints over the trip departure and completion times. The durations of travel between locations are encoded as conditional constraints, and their bounds, as well as guard assignments, are presented in Table 1.1.

Unfortunately, due to the long travel times to and from the candidate Chinese restaurants, no solution can be found that meets all temporal requirements. In this

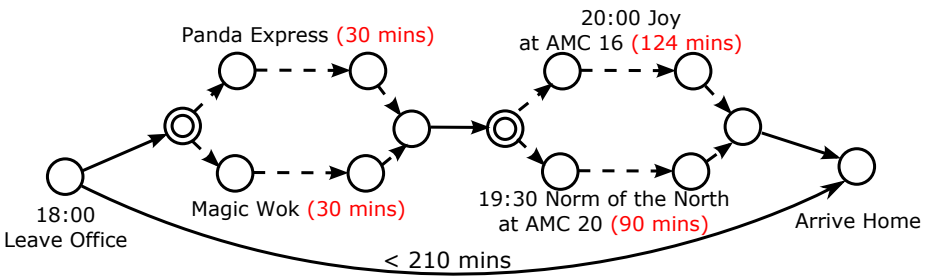


Figure 1-2: A TPN for Simon's dinner and movie activities (the double circles indicate choices between alternative activities)

Traversal Durations	**Guard Assignments**
Office → PE: [40,65]	$Dinner \leftarrow PE$
Office → MW: [30,35]	$Dinner \leftarrow MW$
PE → AMC 16: [25,45]	$Dinner \leftarrow PE$, $Movie \leftarrow JY$
PE → AMC 20: [35,55]	$Dinner \leftarrow PE$, $Movie \leftarrow NN$
MW → AMC 16: [35,45]	$Dinner \leftarrow MW$, $Movie \leftarrow JY$
MW → AMC 20: [40,55]	$Dinner \leftarrow MW$, $Movie \leftarrow NN$
AMC 16 → Home: [25,30]	$Movie \leftarrow JY$
AMC 20 → Home: [20,25]	$Movie \leftarrow NN$
PE for Panda Express, MW for Magic Wok, JY for movie Joy, NN for movie Norm of the North.	

Table 1.1: Travel times between locations (in minutes)

situation, the decision assistant (DA) initiates a discussion with Simon about possible resolutions for his problem.

DA: *You may have dinner at Magic Wok then watch the 8pm Joy at AMC 16. However, due to the length of the movie you won't be back home until 10:34pm. Is that OK?*

Simon: *No, I cannot arrive later than 10pm.*

DA: *OK, then can you leave office 30 minutes earlier? If so you may watch Norm of the North at 7:30pm, and arrive home at 9:30pm.*

Simon: *Sounds good. Thank you.*

Before relaxing any constraints, there is no consistent solution to Simon's plan: three hours and 30 minutes is not enough for him to complete both dinner and movie tasks. Instead of dropping either movie or dinner activity, BCDR tried several options of weakening the constraints that are in conflict, and provided a balanced trade-off for Simon that also respects the new requirements he added during the conversation.

This example demonstrates the advantage of continuous relaxation: it minimizes perturbation to the requirements in the original plan. Compared to discrete relaxations used by prior approaches, which may ask Simon not to watch movie or have dinner, continuous relaxations preserve more of the original problem while restoring consistency. In addition, as a benefit of using the conflict-directed relaxation framework, BCDR is able to adapt to newly added constraints and enumerate relaxations accordingly. When evaluated empirically on a range of travel planning, autonomous underwater vehicle management, and transit network scheduling problems, the continuous relaxation approach demonstrates a substantial improvement in solution quality compared to previous discrete approaches. In addition to solving over-subscribed temporal plans, the continuous relaxation approach has also found applications for feasible ones: given a solution for a temporally feasible plan, it can efficiently find the boundary between consistency and inconsistency, allowing the users to evaluate the amount of redundancy that can be built into a plan.

1.3 Rick-bounded Relaxation Under Temporal Uncertainty

The third contribution of this thesis is an approach for learning conflicts from problems that involve temporal uncertainty, and computing relaxations for them with bounded risk. Uncertainty is commonly encountered in scheduling and planning problems, and is a major cause of over-subscription. Prior work for over-subscribed temporal and planning problems, including the first version of BCDR algorithm (Yu & Williams, 2013), only work with temporal constraints that have controllable bounds whose outcomes can be fully controlled. When applied to problems with uncertain durations, these algorithms may only satisfy a subset of the random outcomes and hence their relaxations may fail during execution.

This is the motivation for us to develop a risk-bounded version of the continuous relaxation approach, which allows BCDR to compute more robust solutions for

real-world scenarios. We first presented this approach as the Conflict-Directed Relaxation with Uncertainty (CDRU) algorithm in (Yu, Fang, & Williams, 2014). This extension to the relaxation framework generalizes the conflict-learning process to use controllability models, instead of consistency. The key innovation of the algorithm is a new conflict learning procedure for strong and dynamic controllability checking, which ensures that a static schedule or a dynamic execution policy for a scheduling problem can be found subject to the uncertain durations.

In addition, the risk-aware approach also supports problems with probabilistic temporal durations and bounded risk of failure. Given a temporal plan with uncertain duration, it proposes execution strategies that operate at acceptable risk levels and pinpoints the source of risk. If no such strategy can be found that meets the risk bound, the risk-aware approach can help us repair the over-subscribed plan by trading off between desirability of solution and acceptable risk levels. The probabilistic extension was first presented in (Yu, Fang, & Williams, 2015), which leverages prior work on probabilistic scheduling (Fang, Yu, & Williams, 2014). The key idea is to diagnose the source of risks by grounding the probabilistic durations to a set-bounded representation through risk allocation, then applying controllability checking and conflict extraction algorithms to identify conflicting constraints from the grounded problem. The extension also introduces the second type of relaxation: **chance constraint relaxation**.

To demonstrate the desired features of BCDR on solving problems with temporal uncertainty, we present an example from the domain of deep-sea explorations. In these missions, correct handling of uncertainties and management of risk are essential requirements for the ocean scientist who makes expedition plans. Imagine that an ocean scientist, Rich, is planning to deploy an autonomous underwater vehicle to survey a volcano eruption on the sea floor. The eruption will occur at around 10:00, following a normal distribution with a variance of 30 minutes. It is 8:00 now, and the vehicle needs to arrive at the site before the start of the eruption. In addition, at least 45 minutes is required for traversing to the site, and 30 minutes for collecting samples. Rich wants the mission to complete in 3 hours, with less than 5% risk of

violating any constraints, such as being late or missing the event. BCDR captures this problem using a chance-constrained probabilistic Temporal Plan Network formulation (cc-pTPN, Figure 1-3). After evaluating all the requirements, the decision assistant determines that no solution exists that meets all requirements. It engages Rich and initiates a discussion to resolve this problem.

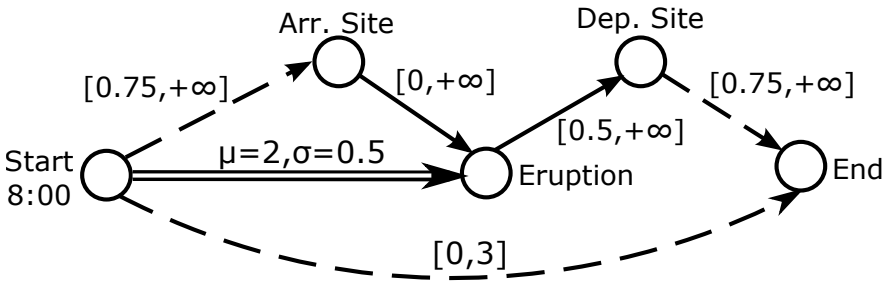


Figure 1-3: The cc-pTPN for the autonomous underwater vehicle's mission (the double arc represents the uncertain duration between start and volcano eruption times)

DA: *I cannot meet all requirements due to the limited mission time and the uncertainty in eruption. Can you* ***extend the mission*** *to 4 hours and 10 minutes.*

Rich: *You can have at most 4 hours for this mission.*

DA: *May I* ***increase the risk bound*** *for this mission to 7.3% in order to meet the duration requirement?*

Rich: *I do not want to take that much risk on this task.*

DA: *Ok, can you* ***shorten the traversal time*** *from the site to the ship by 6 minutes? My plan can then cover 95% of the possible eruption time, between 8:45 and 10:51.*

Rich: *That's fine. Thanks.*

The decision assistant supported by BCDR is able to compute risk-bounded relaxations by making trade-offs between relaxations over both chance and temporal constraints. During the process, it continues to work collaboratively with the user, proposing alternatives and learning new requirements, in order to find better resolutions. Empirical test results on a range of trip scheduling problems show that the risk-bounded relaxation approach generates robust solutions, and is efficient enough

for resolving large-scale plans with temporal uncertainty.

1.4 Domain Relaxations of Parameterized Variables

The final contribution of this thesis is a novel method for computing domain relaxations for resolving conflicts in over-subscribed temporal plans. Recall that our problem is to relax a TPN so that its choices and activities can be grounded and then executed. Further, grounding involves finding a schedule and assignment to the choices. Thus far we have focused on relaxing temporal and risk-bound requirements. We can also achieve consistency by relaxing the domains of the parameters. For example, given that no Chinese restaurant can meet Simon's requirement of arriving home by 9:30pm, it might be better to consider going to a Korean restaurant that is closer to his home. This motivates an alternative approach for relaxation, which can not only suggest relaxing timing requirements, but also recommend good alternative options not encoded in the original domain, such as destinations, in over-subscribed situations.

The extension, called *Domain Relaxation*, works together with temporal relaxation to enable such a capability. Instead of widening the range of temporal bounds, domain relaxation widens the domain of variables. As a result, more options can be considered in the plan, which gives the users more flexibility in resolving their over-subscribed plans.

We introduced the domain relaxation extension to BCDR in (Yu, Shen, Yeh, & Williams, 2016a), which also includes extending the temporal plan formulation with domain constraints for discrete variables. These domain constraints encode the meaning of the domain values of their variables.And domain relaxation resolves conflicts by allowing additional values in the variable domains through weakening their domain constraints. The relaxation of domain constraints are guided by a semantic similarity model generated by the Word2Vec (Mikolov, Sutskever, Chen, Corrado, & Dean, 2013a) and SemanticMemory (Raiman, 2016) packages, which use high-dimension vector representations of concepts trained on a large corpus of

Natural Language data. The word vector approach has been shown to carry semantic meanings (Mikolov, Yih, & Zweig, 2013b) when comparing concepts.

For example, we consider the aforementioned travel example again, but this time Simon cannot compromise on either an early departure from office, or a late arrival at home. In this situation, the decision assistant engages Simon and initiates a discussion about possible resolutions along a different direction.

DA: *You may have dinner at Magic Wok then watch the 8pm Joy at AMC 16. However, due to the length of the movie you won't be back home until 10:34pm. Is that OK?*

Simon: *No, I cannot arrive later than 9:30pm.*

DA: *OK, then can you leave office 30 minutes earlier? If so you may watch Norm of the North at 7:30pm, and arrive home at 9:30pm.*

Simon: *No I cannot leave office before 6pm.*

DA: ***How about eating at Sunny Bowl, a Korean restaurant?*** *It is closer and you can make the 7:30pm movie without leaving any earlier.*

Simon: *Sounds good. Thank you.*

In this example, Simon cannot change the departure and arrival times for his travel plan, hence he rejected the first two proposals. Domain relaxation is able to find Simon a satisfactory solution: it weakened the domain constraints for the restaurant task, such that three new options became available for his trip. In this case, the decision assistant discovered a close alternative, Korean, for the cuisine requirement of restaurant. It then queried the knowledge base to retrieve additional restaurant candidates, and found one that is closer to Simon's route and satisfies all temporal constraints (Figure 1-4).

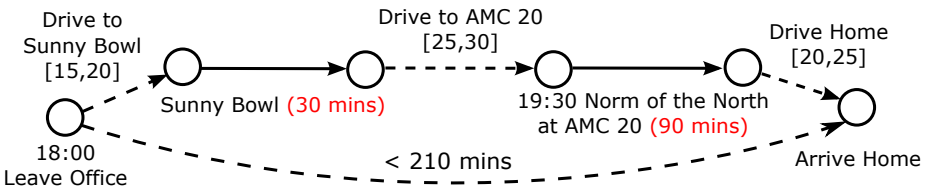


Figure 1-4: A solution for Simon's problem enabled by relaxing the cuisine requirement

This example demonstrates the desired behavior of domain relaxation: resolving conflicts in over-subscribed problems by **weakening** the domain constraints, which enables more options to be considered that are not encoded in the original problem. Given a conflict, BCDR has three options to resolve it: (1) continuous relaxations for temporal bounds; (2) continuous relaxations for the risk bound; and (3) domain relaxations for discrete variables. Through resolving conflicts by computing all three types of relaxations simultaneously, BCDR is able to enumerate them in best-first order while resolving over-subscribed temporal plans. When evaluated empirically on a range of urban trip planning scenarios, the domain relaxation approach demonstrates a substantial improvement in flexibility compared to the temporal relaxation only approach. It enables solutions to be found in many scenarios that were previously infeasible, or finds more preferred ones by avoiding drastic relaxations for temporal constraints.

1.5 Summary of Contributions and Conclusions

In this section we give a summary of the contributions of this thesis, and the major conclusions drawn.

1. A Conflict-directed Relaxation Framework

   We introduce a conflict-directed relaxation framework, Best-first Conflict-Directed Relaxation, which has three key properties: (a) detecting conflicts in over-subscribed temporal plans; (b) enumerating relaxations for resolving conflicts in best-first order; and (c) incorporating user feedback to dynamically improve solutions.

2. Continuous Relaxations for Temporal Constraints

   We present an extension to the tester and candidate generator of BCDR that (a) encodes the cause of failure as hybrid conflicts; and (b) computes preferred continuous relaxations, instead of discrete relaxations, for resolving these conflicts.

3. Risk-bounded Relaxations Under Temporal Uncertainty

   We present the second extension to BCDR for handling temporal uncertainty, which (a) learns conflicts using strong and dynamic controllability models; and (b) computes risk-bounded temporal relaxations subject to chance constraints.

4. Domain Relaxation for Parameterized Variables

   Finally, we introduce the extension to the relaxation generator of the conflict-directed relaxation framework that (a) computes domain relaxations, instead of temporal relaxations, for conflicts using semantically similar alternatives; and (b) simultaneously enumerates preferred temporal and domain relaxations for hybrid conflicts in temporal plans.

1.6 Organization of Thesis

First, in Chapter 2 we present the definition for over-subscribed temporal plans and their relaxations based on the Temporal Plan Network formalism. It includes the necessary background, problem statement and notations that are used throughout the dissertation.In Chapter 3, we introduce the Best-first Conflict-Directed Relaxation algorithm, and its application for computing preferred discrete relaxations, which are the foundation of all the relaxation techniques presented in this thesis.

Then in Chapter 4, we present the extension to BCDR for computing continuous temporal relaxations, and how it interleaves the enumeration of both discrete and continuous relaxations. In Chapter 5, we describe the augmented continuous relaxation extension for supporting risk-bounded temporal relaxations.

In Chapter 6, we present the final extension to BCDR for computing domain relaxations, and how it enumerates both temporal and domain relaxations simultaneously. In Chapter 7, we present the design and implementation of the advisory system, Uhura, for collaborative diagnosis of over-subscribed temporal plans. This chapter describes how Uhrua integrates the BCDR algorithm, a conversational user interface and point-of-interest database, and discusses its applications in the domain

of urban travel planning. In Chapter 8, we present the results from empirical evaluations of the different relaxation techniques. Finally, we conclude in Chapter 9 and discuss future research directions.

Chapter 2

Problem Statement

In this chapter, we present the problem statement for the BCDR algorithm, which includes the over-subscribed temporal plans, and the four types of relaxations, discrete, continuous, risk-bounded and domain relaxations, that can be used to resolve conflicts in them. Throughout this chapter, we will be using two examples from urban travel planning and deep-sea exploration to illustrate the different elements in the inputs and outputs of BCDR.

2.1 Temporal Plan Network

The BCDR algorithm takes a Temporal Plan Network (TPN) as input, and produces a plan, as well as suspension of some temporal constraints if necessary. The definition of TPN was first presented in (Kim, Williams, & Abramson, 2001) as a compact representation of multiple alternative plans, which captures the activities, choices as well as the temporal constraints. Graphically, a TPN can be represented by a node-edge graph, similar to the Progress Evaluation and Review Technique chart (Stauber, Douty, Fazar, Jordan, Weinfeld, & Manvel, 1959). Each node represents an **event** in the plan, which is a unique point of time. Each arc represents an **episode**, which can be either an activity, or a temporal constraint that restricts the duration between time points. Formally, we define a TPN as the following:

Definition 1. *A TPN is an 7-tuple* $\langle P, Q, V, E, L_e, L_p, f_p \rangle$ *where:*

- *P is a set of controllable finite domain discrete variables;*
- *Q is the collection of domain assignments to P;*
- *V is a set of events representing designated time points;*
- *E is a set of episodes between pairs of events $v_i, v_j \in V$;*
- *$L_e : E \rightarrow 2^Q$ is a guard function that attaches conjunctions of assignments in Q, $q_i \in Q$, to some episodes $e_i \in E$;*
- *$L_p : P \rightarrow 2^Q$ is a guard function that attaches conjunctions of assignments in Q, $q_i \in Q$, to some discrete variable $p_i \in P$;*
- *$f_p : Q \rightarrow \mathcal{R}^+$ is a function that maps each assignment to every controllable discrete variable, $q_i \in Q$, to a positive **reward** value;*

Function L_e and L_p specify the guard assignments of some episodes in E and variables in P. If the guard assignments of a conditional variable are not satisfied, then it is deactivated and will not be assigned. If the guard assignments for an episode are not satisfied, then the episode's temporal requirements do not need to be satisfied. For example, in Simon's trip, if he chooses to watch movie Joy instead of Norm of the North, then the activities of going to AMC 20 as well as the temporal constraint associated with the duration of the movie do not need to be satisfied.

In addition to the TPN, we also include two additional inputs for BCDR in order to model the temporal plan: (1) a set of agent models that encode their requirements, preferences and mode of travel; and (2) a routing function for estimating the travel times between locations for each agent.

Definition 2. *Each agent model is a 6-tuple $\langle P_i, E_i, Q_i, f_{pi}, OD_i, m \rangle$, where:*

- *$P_i \subseteq P$ is a set of finite domain variables representing decisions for this agent i;*
- *$Q_i \subseteq Q$ is the collection of domain assignments to P_i;*
- *$E_i \subseteq E$ is a set of episodes representing activities and requirements for this agent i;*

- $f_{pi} : Q_i \rightarrow \mathcal{R}^+$ *is a function that maps each assignment to discrete variable,* $q_{ij} : p_i \leftarrow value_j$*, to a positive* ***reward****;*

- OD_i *is a pair of latitude/longitude locations that specifies the origin and destination of agent* i*;*

- $m_i \in \langle driving, biking, walking \rangle$ *is the mode of travel for agent* i*.*

For an agent i, the model defines the subset of goals and requirements in the TPN that belongs to the agent. BCDR uses the model to ensure that its solution is consistent for each agent, such that their plans originates from their original location, and ends at the specified destination. In addition, when a solution is generated, the agent model allows BCDR (and Uhura supported by it) to decide which part of the solution should be presented to which agent, and provide more compact explanations to the users. It is possible that one decision variable or episode is shared among multiple agents. In this case all of them will be informed when a choice or relaxation is made.

Next, we define the routing function as the following:

Definition 3. *The routing function* $f_r : (O, D, m) \rightarrow \langle lb, ub, pl \rangle$ *takes in three parameters as input, and outputs a 3-tuple, where:*

- $O = \langle latitude, longitude \rangle$ *is a coordinate encoding the origin of the route;*

- $D = \langle latitude, longitude \rangle$ *is a coordinate encoding the destination of the route;*

- $m \in \langle driving, biking, walking \rangle$ *indicate the mode of travel;*

- $lb \in \mathcal{R}^+$ *is the lower bound on the duration of travel;*

- $ub \in \mathcal{R}^+$ *is the upper bound on the duration of travel.*

- $pl = \langle latitude1, longitude1 \rangle, \langle latitude2, longitude2 \rangle, ...$ *is an array of coordinates encoding the polyline of the route.*

For the travel between any pair of locations, the routing function computes the fastest route between them and outputs the travel times, given the mode of travel available to the agent. The route is encoded by the polyline, which is an ordered set of coordinates. The travel time is encoded using a set-bounded duration, whose lower and upper bounds represents the best and worst estimates given traffic conditions. Depending on the user requirements, BCDR has multiple ways to handle these bounds while checking the temporal feasibility of a candidate solution or resolving conflicting constraints that involve uncertainty. For example, an optimistic approach may only considers the lower bounds of these traversal durations, while a more robust approach considers them as random variables and a solution must be robust to all possible outcomes. We will elaborate on these different treatments when discussing the temporal relaxation algorithms in Chapter 5.

The solution to a TPN is a 3-tuple, which defines a temporally feasible set of choices to the variables, and set of activities for each agent specified in the input TPN.

Definition 4. *The solution to a TPN is a 3-tuple $\langle A, S, E' \rangle$, where:*

- *A is a complete set of assignments to variables in P;*
- *S is a set of additional assignments that defines ordering over activities in the TPN;*
- *E' is a set of episodes that encodes the traversal activities between locations, generated by the routing function. Each $e' \in E'$ encodes the traversal time associated with an agent's movement between locations specified in A, following the order defined by S.*

As presented earlier, BCDR may also present the solution on an agent-by-agent basis, such that the individual solution only contains choices and relaxations associated with that agent. The agent-based solution is also a 3-tuple $\langle A_i, S_i, E'_i \rangle$, which encodes the choices and relaxations associated with the variables and episodes supplied by agent i.

2.2 Discrete Relaxations

2.2.1 Example Scenario

Consider a scenario in which a user, Simon, is planning for an evening outing trip with Uhura, our plan assistant that builds on BCDR. Simon plans to leave the office at 6pm, and would like to have dinner with a friend, Christian, at a nice Chinese restaurant. He also plans to watch a comedy movie this evening. In addition, Simon needs to be home before 9:30pm. Uhura's task is to work out a robust plan with them, which includes the choices for restaurant and movie showing, the sequence of these activities, and appropriate adjustments to the timing requirements, if necessary. In this section, we describe informally how Uhura collaborates with Simon and Christian, and resolves their over-subscribed travel plan.

Uhura starts with the TPN for Simon and Christian's travel problem (Figure 2-1), which encodes the two activities for dinner and movie, as well as the temporal constraints over the trip departure and completion times. The two activities requested by them, *watch a comedy movie* and *dine at a Chinese restaurant*, are highlighted in bold. There are four dotted arcs connecting the activities to the beginning and end of the trip, which represent temporal constraints that encode the sequencing requirement for the activities: they must take place after leaving office, and finish before arriving home. Finally, there is one dotted arc connecting the first and last events, which encodes the overall temporal constraint of 210 minutes (from 6pm to 9:30pm) for the entire trip. Each activity and temporal requirement is also tagged with SI (Simon) or CR (Christian) to indicate the participation or sources for them.

Given the variables for the activities, we can pass their description to a knowledge base, which can search through multiple data sources and retrieve candidate options for them. We will explain this in detail how the retrieval process works in Chapter 7. These options are then encoded as alternative episodes for the activities and added to the TPN. For example, the expanded TPN for Simon and Christian's outing trip is shown in Figure 2-2. The comedy movie activity is replaced by two movie showings at different theaters, while the Chinese restaurant activity is replaced by

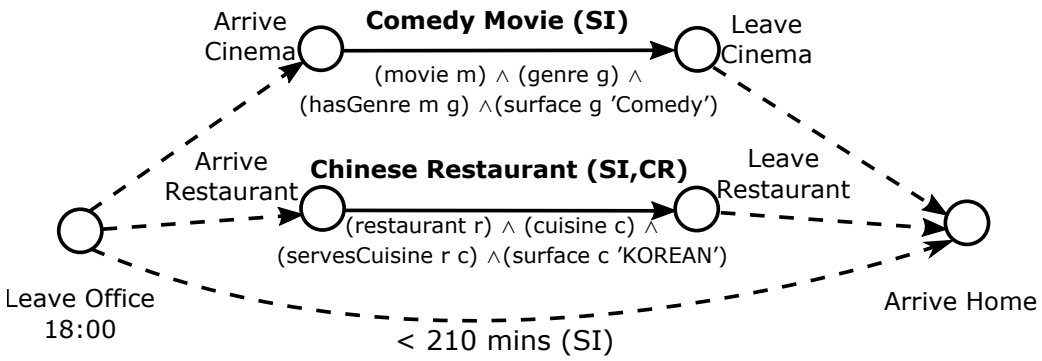


Figure 2-1: A TPN for Simon and Christian's trip

two restaurant options. In the TPN graph, these grounded options for each activity share one common start event, which is represented by a double circle and indicates that the subsequent episodes are alternatives. In addition, each grounded activity is associated with a duration (highlighted in the figure). These durations encode the length of the movie or dinner. The constraints for traversals between locations (Table 2.1) are omitted from the graph to save space.

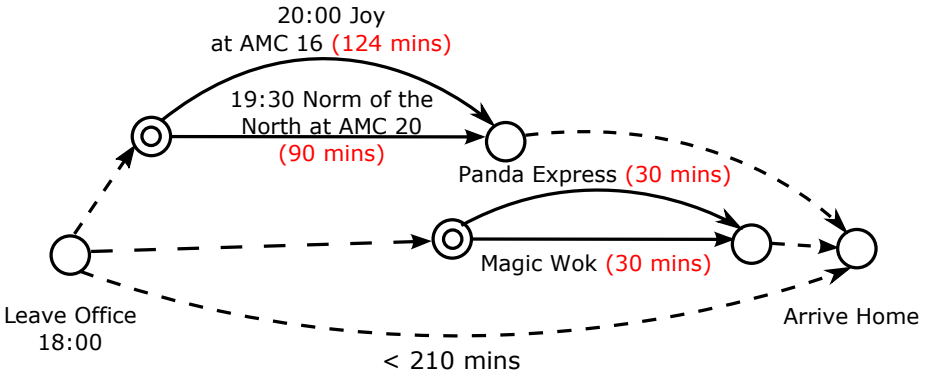


Figure 2-2: An expanded TPN with activity candidates

Next, Uhura passes the expanded TPN to BCDR to fill in the details of the plan, select the candidate for each activity, and determine their orders while meeting all the requirements. The result is a complete plan with grounded activities for both Simon and Christian. If no feasible plan can be found that meets all requirements, as in this example, BCDR will try relaxing the temporal bounds of some episodes in order to make room for completing all activities. This is the situation encountered by Simon in this example: due to the long travel times to and from the candidate

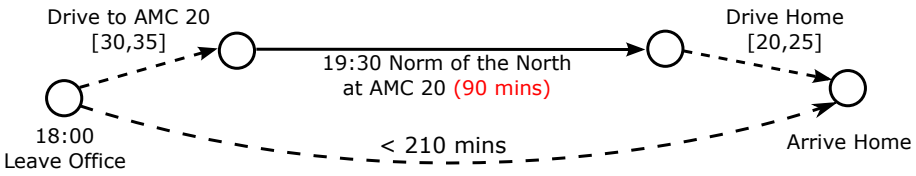
Traversal Durations	
Office → PE: [40,65]	PE → Home: [25,30]
Office → MW: [30,35]	MW → Home: [20,25]
Office → AMC 16: [40,65]	AMC 16 → Home: [25,30]
Office → AMC 20: [30,35]	AMC 20 → Home: [20,25]
PE → AMC 16: [25,45]	AMC 16 → PE : [25,45]
PE → AMC 20: [35,55]	AMC 20 → PE: [35,55]
MW → AMC 16: [35,45]	AMC 16 → MW: [35,45]
MW → AMC 20: [40,55]	AMC 20 → MW: [40,55]

Table 2.1: Travel times between locations (PE stands for Panda Express, and MW stands for Magic Wok)

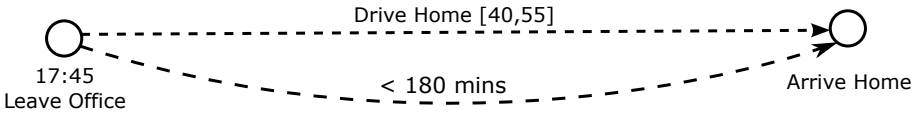
Chinese restaurants, no solution can be found that meets all temporal requirements. Hence Uhura engages Simon and Christian, and initiates a discussion about possible resolutions for his problem.

> **Uhura**: *Simon, you may have dinner at Magic Wok then watch the 8pm Joy at AMC 16. However, due to the length of the movie you have to remove the constraint on the time of arriving home. Is that OK?*
>
> **Simon**: *No, I cannot delay my arrival time.*
>
> **Uhura**: *OK, then Simon can you remove the constraint on departure time from Office? If so you may watch Norm of the North at 7:30pm, and arrive home on time.*
>
> **Simon**: *No I cannot leave office before 6pm.*
>
> **Uhura**: *Simon and Christian, How about not having dinner tonight?*
>
> **Christian**: *That's fine.*
>
> **Simon**: *Sounds good. Thank you.*

In this example, BCDR proposed different relaxations that resolves the conflicts in the Simon and Christian's plan. Since they cannot suspend the constraints on the departure and arrival times, the first two proposals were rejected. BCDR incorporates their inputs on earlier solutions, and kept proposing new ones that respects all their feedback until an agreement is reached, which enables a feasible plan for both Simon and Christian (Figure 2-3).



(a) Simon's trip with suspended dinner episode



(b) Christian's trip with suspended dinner episode

Figure 2-3: A solution enabled by relaxed cuisine constraint

This example demonstrates the desired features of BCDR on resolving over-subscribed temporal plans using discrete relaxations: it allows Uhura to work collaboratively with the users to resolve conflicts in over-subscribed plans, enumerating discrete relaxations in best-first order and providing rationale for the relaxations made.

2.2.2 Definitions

We extend the definition of TPN with an additional set of relaxation episodes, RE, to support discrete relaxations. For its solutions, we also include a set of suspended episodes from RE that are necessary for making the solution consistent. Formally, we define a relaxable TPN as the following:

Definition 5. *A relaxable TPN is a 9-tuple* $\langle P, Q, V, E, RE, L_e, L_p, f_p, f_e \rangle$, *where:*

- P *is a set of controllable finite domain discrete variables;*
- Q *is the collection of domain assignments to* P*;*
- V *is a set of events representing designated time points;*
- E *is a set of episodes between pairs of events* $v_i, v_j \in V$*;*
- $RE \subseteq E$ *is a set of relaxable episodes that can be suspended;*

- $L_e : E \rightarrow 2^Q$ *is a guard function that attaches conjunctions of assignments in* Q, $q_i \in Q$, *to some episodes* $e_i \in E$;
- $L_p : P \rightarrow 2^Q$ *is a guard function that attaches conjunctions of assignments in* Q, $q_i \in Q$, *to some discrete variable* $p_i \in P$;
- $f_p : Q \rightarrow \mathcal{R}^+$ *is a function that maps each assignment to every controllable discrete variable,* $q_i \in Q$, *to a positive* ***reward*** *value;*
- $f_e : RE \rightarrow \mathcal{R}^+$ *is a function that maps the suspension to one relaxable temporal constraint* $e_i \in RE$, *to a positive* ***cost*** *value.*

The solution to a relaxable TPN is a set of assignments to variables in P such that all activated episodes that are not suspended (not in R_e) are temporally consistent.

Definition 6. *The solution to a relaxable TPN is a 4-tuple* $\langle A, S, R_e, E' \rangle$, *where:*

- A *is a complete set of assignments to variables in* P;
- S *is a set of additional assignments that defines ordering over activities in the TPN;*
- R_e *is a set of episodes in RE that are suspended.*
- E' *is a set of episodes that encodes the traversal activities between locations, generated by the routing function. Each* $e' \in E'$ *encodes the traversal time associated with an agent's movement between locations specified in* A, *following the order defined by* S.

In addition, given a solution, we may also define if its set of discrete relaxations are **minimal**. A set of discrete relaxations of a temporally consistent solution is minimal if and only if none of its strict subset still makes the solution temporally consistent:

Definition 7. *A solution* $\langle A, S, R_e, E' \rangle$ *has a minimal set of discrete relaxation if and only if:*

- $\langle A, S, R_e, E' \rangle$ *is a temporally consistent solution;*
- $\langle A, S, R'_e, E' \rangle$, *where* $R'_e \subset R_e$ *is not temporally consistent.*

2.3 Continuous Relaxations

2.3.1 Example Scenario

For continuous relaxations, we use an example from the deep-sea exploration scenario to demonstrate its definitions. Consider the following example in which an ocean scientist, Rich, is planning to deploy an autonomous underwater vehicle (AUV), Sentry (Figure 2-4b), to survey the sea floor close to the coast of Northern California. This mission is expected to start at 11:00AM from the R/V Atlantis (Figure 2-4a), and Rich has reserved the vehicle until 2:00PM. During this mission, he would like to visit one of the two asphalt mounds (Figure 2-4c). The two sites are denoted by Location A and B. Rich has a preference over the two options and their required survey times vary from 35 minutes to 50 minutes. After visiting the mound, Rich wants to scan one of the three nearby methane seeps (Figure 2-4d), denoted by Location X, Y or Z. It takes a different amount of time at each site due to their sizes and complexity. Finally, traversal times to and between these sites are different, and Sentry must return to the ship in three hours (11:00AM to 2:00PM) so that the next scientist can start their mission on time.

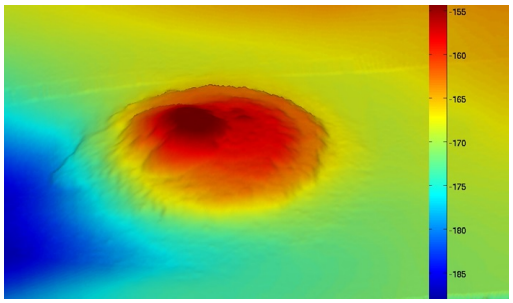
We again model Sentry's mission using a TPN, which includes: which asphalt mounds site to survey, which methane seep site to scan, how much time to spend at each site, and whether to extend the mission length. We start by defining two variables for the choices he needs to make: AM (asphalt mounds sites) and MS (methane seep sites). AM has two options in its domain: A (40) and B (100). Each option is associated with a positive reward value that represents Rich's preference towards it, the larger the better. The other variable MS for methane seeps sites has three options: X (73), Y (80), and Z (47).

Next, we define twelve events in the plan (Table 2.2): a reference point in time (S) that represents the beginning of the trip at 11am; a time point that indicates the end of the trip (E); and time points representing the arrival and departure of each location (asphalt mounds sites A and B, methane seep sites X, Y, and Z).

Table 2.3 shows all the conditional episodes in the TPN that encode the temporal

(a) Research Vessel, Atlantis

(b) Autonomous Underwater Vehicle, Sentry



(c) Sonar data of an undersea asphalt mound collected by Sentry

(d) An active methane seep in South Ellwood Oil Field (Photo by R K Nelson)

Figure 2-4: Vehicles and survey targets of the example expedition (Courtesy WHOI)

Events			
Mission starts	S	asphalt mounds site A arrive/leave	A_A,A_L
Mission ends	E	asphalt mounds site B arrive/leave	B_A,B_L
methane seep site X arrive/leave		X_A,X_L	
methane seep site Y arrive/leave		Y_A,Y_L	
methane seep site Z arrive/leave		Z_A,Z_L	

Table 2.2: Events in Sentry's mission TPN

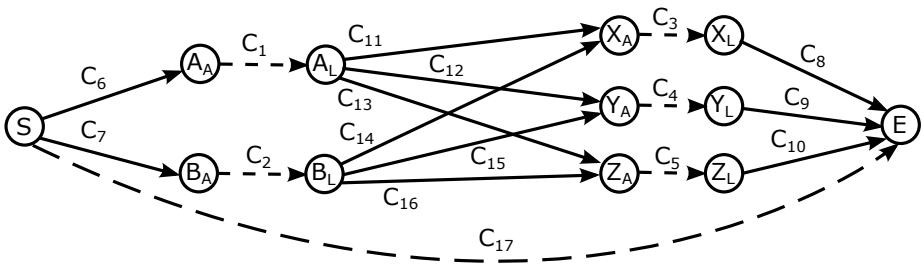


Figure 2-5: A graphical representation of Rich's mission TPN

relaxations between events. A subset of these episodes only hold for some choices, such as the duration of survey at site A (C_1) for assignment $AM \leftarrow A$. We represent this by saying that episode C_1 is $ACTIVE$ only when the choice $AM \leftarrow A$ is made,

Constraints (in minutes)		
C_1(R):A_L-$A_A \in [50, 60]$	C_6:A_A-$S \in [45, 65]$	$AM \leftarrow A$
C_2(R):B_L-$B_A \in [45, 60]$	C_7:B_A-$S \in [30, 50]$	$AM \leftarrow B$
C_3(R):X_L-$X_A \geq 60$	C_8:E-$X_L \in [28, 35]$	$MS \leftarrow X$
C_4(R):Y_L-$Y_A \geq 65$	C_9:E-$Y_L \in [30, 32]$	$MS \leftarrow Y$
C_5(R):Z_L-$Z_A \geq 100$	C_{10}:E-$Z_L \in [50, 60]$	$MS \leftarrow Z$
C_{11}:X_A-$A_L \in [51, 54]$	$AM \leftarrow A$ and $MS \leftarrow X$	
C_{12}:Y_A-$A_L \in [42, 45]$	$AM \leftarrow A$ and $MS \leftarrow Y$	
C_{13}:Z_A-$A_L \in [30, 55]$	$AM \leftarrow A$ and $MS \leftarrow Z$	
C_{14}:X_A-$B_L \in [22, 24]$	$AM \leftarrow B$ and $MS \leftarrow X$	
C_{15}:Y_A-$B_L \in [21, 25]$	$AM \leftarrow B$ and $MS \leftarrow Y$	
C_{16}:Z_A-$B_L \in [30, 35]$	$AM \leftarrow B$ and $MS \leftarrow Z$	
C_{17}(R):E-$S \in [0, 180]$		

Table 2.3: Episodes in the TPN of Sentry's mission (Solid arrows represent traversal durations, while dotted arrows represent Rich's temporal requirements)

and indicate this by labeling the episode with $AM \leftarrow A$. The episode is called conditional episode, and the label is called the guard of the episode. In this problem, episodes C_1 through C_5 have inequality temporal constraints that represent Rich's desired length of survey at five locations. For example, B_L–$B_A \geq 35$ indicates that Rich would like Sentry to spend at least 35 minutes at asphalt mound site B. These episodes are labeled by the assignments made to the decision variables: an episode is activated only if its label assignment is made. For example, C_2 will be considered only if Rich chooses to visit site B, as shown in the right side of Table 2.3. Episodes C_6 through C_{16} encode simple temporal constraints that model the traversal time required between locations. They are conditioned on assignments made to either AM or MS, or both (C_{11} through C_{16}). Finally, C_{17} constrains the duration of Sentry's mission to three hours. The TPN can be visualized using a directed graph (Figure 2-5), in which nodes represent events and arcs represent episodes.

Some of the episodes followed by a symbol 'R' (also highlighted in dotted arcs in the graph: C_1 through C_5 and C_{17}) are continuously relaxable episodes. Their lower and/or upper bounds can be relaxed in order to restore the temporal feasibility of the plan, if necessary. Each relaxable episode comes with one or two cost functions that describe Rich's preferences towards the weakening for their upper and lower bounds.

These functions map the relaxation from LB to LB', or from UB to UB', to a positive cost value, as seen in Figure 2-6. If the upper bound of C_{17} is increased from 180 minutes to 200 minutes, meaning that Rich extends his mission by 20 minutes, the cost will be 40. On the other hand, if he shortens the survey time by reducing the lower bound of C_3 to 40, the cost would be 80. In this example, we assume that all other relaxable episodes have linear cost functions with gradient 1 for simplicity, but the approach is generalizable to arbitrary monotonic functions (decreasing for lower bounds, and increasing for upper bounds).

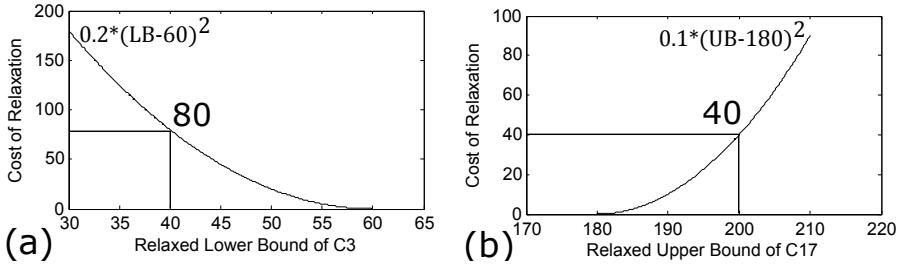


Figure 2-6: Preference functions for C_3 and C_{17}

Relaxation 1		**Relaxation 2**		**Relaxation 3**
$AM \leftarrow B$ $MS \leftarrow Y$ C_2 to $\in [39, 60]$ C_{17} to $\in [0, 185]$ Utility: 171.5	**Do not relax** C_{17}	$AM \leftarrow B$ $MS \leftarrow X$ C_3 to ≥ 57.5 C_2 to $\in [42.5, 60]$ Utility: 169.25	C_2 **is at least 44**	$AM \leftarrow B$ $MS \leftarrow Y$ C_4 to ≥ 55 C_2 to $\in [44, 60]$ Utility: 169

Table 2.4: Three preferred continuous relaxations to Rich's TPN

Before making any relaxations, there is no temporally consistent solution to the problem. The cause of failure is that three hours is not enough for Sentry to complete both activities: traversing to the nearest asphalt mounds and methane seep site will consume at least 80 minutes, which brings the minimum trip duration to nearly 190 minutes. Therefore, one or more episodes' temporal constraints need to be relaxed.

Table 2.4 shows three continuous relaxations for the TPN ranked in best-first order. Relaxation 1, suggests visiting site B and Y. The survey time for B should be reduced to 39 minutes and the mission should be extended by 5 minutes. The utility of the relaxation is 171.5, which is computed by summing up the reward of

two assignments, $AM \leftarrow B$ and $MS \leftarrow Y$, and subtracting the cost of relaxing C_2 and C_{17}. If Rich decides not to relax C_{17}, Relaxation 2 will be preferred since it only takes Sentry to methane seeps site X, shortens the scan time to 57.5 minutes and reduces the survey time at B to 42.5 minutes. If Rich requires that survey time at the asphalt mound site should be no less than 44 minutes, Relaxation 3 will be more preferred since it respects both additional requirements.

2.3.2 Definitions

The AUV mission example illustrates the modeling of continuous relaxations for over-subscribed temporal plans, and demonstrates the most significant advantage of continuous relaxation: it weakens the temporal requirements to the minimal extent. Compared to discrete relaxations, which may ask Rich to give up on the survey for asphalt mound sites or methane seeps, continuous relaxations preserve more of the original plan while restoring its temporal feasibility. In this subsection, we define the extensions to the TPN formulation and its solutions for supporting continuous relaxations.

Definition 8. *A continuously relaxable TPN is a 9-tuple* $\langle P, Q, V, E, RE, L_e, L_p, f_p, f_e \rangle$, *where:*

- P *is a set of controllable finite domain discrete variables;*
- Q *is the collection of domain assignments to* P*;*
- V *is a set of events representing designated time points;*
- E *is a set of episodes between pairs of events* $v_i \in V$*;*
- $RE \subseteq E$ *is a set of relaxable episodes whose temporal bounds can be continuously relaxed;*
- $L_e : E \rightarrow 2^Q$ *is a guard function that attaches conjunctions of assignments in* Q, $q_i \in Q$, *to some episodes* $e_i \in E$*;*

- $L_p : P \rightarrow 2^Q$ *is a guard function that attaches conjunctions of assignments in* Q, $q_i \in Q$, *to some discrete variable* $p_i \in P$;

- $f_p : Q \rightarrow \mathcal{R}^+$ *is a function that maps each assignment to every controllable discrete variable,* $q_i \in Q$, *to a positive* ***reward*** *value;*

- $f_e : (e_i, e_i') \rightarrow \mathcal{R}^+$ *is a function that maps the relaxation to one relaxable episode* $e_i \in RE$, *from* e_i *to* e_i', *to a positive* ***cost*** *value.*

Here we refer to the deep-sea exploration scenario as a grounded example for the definition. In the TPN model of the scenario, P contains two finite domain variables AM and MS. Their domain values Q contains two sets: $\{A, B\}$ for AM and $\{X, Y, Z\}$ for MS. The set V contains all events in Table 2.2, while the set E contains all episodes in Table 2.3. The relaxable episodes, RE, consists of C_1 through C_5 and C_{17}. Guard function L_e attaches assignments of AM and MS to conditional episodes in E, which are C_1 through C_{16} in this example. Finally, the assignment reward function, f_p, is defined over the five variable assignments and associates them with positive real values: $AM \leftarrow A$ (40), $AM \leftarrow B$ (100), $MS \leftarrow X$ (73), $MS \leftarrow Y$ (80) and $MS \leftarrow Z$ (47). While the relaxation cost function, f_e, is defined over the six episodes in RE, as shown in Figure 2-6.

To allow the continuous relaxation for an over-subscribed temporal plan, we include relaxable temporal episodes in the extended definition of TPN (RE), similar to the soft constraints in a Simple Temporal Problem with Preferences (Rossi et al., 2002). We do not use a disjunctive set of temporal bounds for soft constraints. Instead, the temporal bounds of an episode can be relaxed **continuously** at the price of increasing cost. The cost is defined over the degree of relaxation made to the lower and upper bounds. Continuous relaxation provides greater flexibility in resolving over-subscribed plans: it does not limit our options to the predefined alternative temporal bounds, and allows us to weaken the constraint to the minimum extent necessary.

Reward function f_p is defined over the assignments to controllable discrete variables $q_i \in Q$. Each assignment is mapped to a positive reward value, such as

$MS \leftarrow X : 73$. The larger the number is, the more preferred the choice will be. Cost function f_e is defined over relaxable episodes. The cost of relaxing an upper bound $E_{ij} : v_j - v_i \leq u_{ij}$ from u_{ij} to u'_{ij} is $f_{eij}(u'_{ij} - u_{ij})$. Figure 2-6b shows an example function defined over $u'_{ij} - u_{ij}$.

The cost function for episodes that restrict the lower bounds between two events is $f_{eij}(l_{ij} - l'_{ij})$. This is illustrated in Figure 2-6a. We assume that the user always prefers smaller relaxations. The motivation of this assumption is that people generally prefer to minimize the perturbations to the original requirements, and penalize larger deviations from them. Therefore, all f_e functions must be monotonically increasing, and equal to 0 when there is no relaxation. f_e can be viewed as a semi-convex (Khatib et al., 2001) function with a segment of zero cost when there is no relaxation. This assumption simplifies our relaxation process, as the tightest relaxation will always result in the lowest cost.

Definition 9. *A solution to a continuously relaxable TPN is a 4-tuple* $\langle A, S, R_e, E' \rangle$ *such that all activated constraints are temporally consistent, where:*

- *A is a complete set of assignments to variables in P;*
- *S is a set of additional assignments that defines ordering over activities in the TPN;*
- *R_e is a set of continuous temporal relaxations for some episodes in RE.*
- *E' is a set of episodes that encodes the traversal activities between locations, generated by the routing function. Each $e' \in E'$ encodes the traversal time associated with an agent's movement between locations specified in A, following the order defined by S.*

where a continuous temporal relaxation is defined as a tuple, $\langle e, r_L, r_U \rangle$, as the following:

- e is an episode in RE;
- r_L is a weakened lower bound for e and $r_L \leq e_{lowerbound}$;

- r_U is a weakened upper bound for e and $r_U \geq e_{upperbound}$.

The utility of a solution is computed by subtracting the relaxation cost from the assignment reward: $\sum_i f_p(p_i \leftarrow value_i) - \sum_i f_e(e_i \rightarrow e_i')$. For example, Solution 1 in Table 2.4 consists of two assignments and two relaxations: assignment $AM \leftarrow B$ and $MS \leftarrow Y$ have reward of 100 and 80, while the cost of relaxing C_2 and C_{17} are 6 and 2.5, respectively. Hence the utility value of this solution is 171.5, which is computed by summing up the rewards, and subtracting the cost of relaxing C_4 and C_{17} from it. Given a TPN, the most preferred relaxation to it is the one with the highest utility value according to f_p and f_e.

Finally, similar to discrete relaxations, we may also define minimal set of continuous relaxations in a solution. A set of continuous relaxations of a temporally consistent solution is minimal if and only if none of its strict subset, or any relaxation with strictly tighter bounds, still makes the solution temporally consistent:

Definition 10. *A solution $\langle A, S, R_e, E' \rangle$ has a minimal set of continuous relaxation if and only if:*

- $\langle A, S, R_e, E' \rangle$ *is a temporally consistent solution;*
- $\langle A, S, R_e', E' \rangle$*, where $R_e' \subset R_e$ is not temporally consistent;*
- $\forall re \in R_e$*, $re' \subset re$ and $R_e \backslash \{re\} \cup \{re'\}$ is not temporally consistent.*

If the cost function f_e is strictly increasing with the extent of relaxation, $|r_L - e_{lowerbound}|$ and $|r_U - e_{upperbound}|$, then the consistent set of continuous relaxations with the lowest cost is guaranteed to be minimal, since any non-minimal set of relaxations must have higher costs.

2.4 Risk-bounded Relaxations

In this section, we present two additional extensions of continuous temporal relaxations, for plans with set-bounded uncertain durations, and probabilistic temporal durations with information on the likelihood of outcomes.

2.4.1 Plans with Set-bounded Uncertain Temporal Durations

Uncertainty is commonly encountered in temporal scheduling and planning problems, and can often lead to over-subscribed situations. In Rich's AUV mission described earlier, the traversal times between locations are often non-deterministic. When applying the deterministic formulation to model such problems, their relaxations may fail since they only satisfy a subset of the possible outcomes for the uncertain durations. Hence, we present an extension to the TPN formulation, called Temporal Plan Network with Uncertainty (TPNU), for modeling relaxation problems with uncertain duration. The definitions of TPN and TPNU differ only in the terms of temporal durations: in addition to episodes with controllable durations, a TPNU may also contain episodes with set-bounded uncertain durations.

We use Rich's mission to illustrate the modeling of continuous relaxations with uncertainty duration. Table 2.5 repeats all the episodes for his mission. Note that episodes C_6 through C_{16} are highlighted in bold: they have uncontrollable temporal durations and encode the traversal times between locations. Their temporal bounds indicate the domain of the random outcomes. Similar to TPN, TPNU can also be visualized using a node-arc graph, in which episodes with uncontrollable durations are represented by double arcs (Figure 2-7).

$C_1(R)$:A_L-$A_A \in [50, 60]$	$\mathbf{C_6}$:A_A-$S \in [45, 65]$	$AM \leftarrow A$
$C_2(R)$:B_L-$B_A \in [45, 60]$	$\mathbf{C_7}$:B_A-$S \in [30, 50]$	$AM \leftarrow B$
$C_3(R)$:X_L-$X_A \geq 60$	$\mathbf{C_8}$:E-$X_L \in [28, 35]$	$MS \leftarrow X$
$C_4(R)$:Y_L-$Y_A \geq 65$	$\mathbf{C_9}$:E-$Y_L \in [30, 32]$	$MS \leftarrow Y$
$C_5(R)$:Z_L-$Z_A \geq 100$	$\mathbf{C_{10}}$:E-$Z_L \in [50, 60]$	$MS \leftarrow Z$
$\mathbf{C_{11}}$:X_A-$A_L \in [51, 54]$	$AM \leftarrow A$ and $MS \leftarrow X$	
$\mathbf{C_{12}}$:Y_A-$A_L \in [42, 45]$	$AM \leftarrow A$ and $MS \leftarrow Y$	
$\mathbf{C_{13}}$:Z_A-$A_L \in [30, 55]$	$AM \leftarrow A$ and $MS \leftarrow Z$	
$\mathbf{C_{14}}$:X_A-$B_L \in [22, 24]$	$AM \leftarrow B$ and $MS \leftarrow X$	
$\mathbf{C_{15}}$:Y_A-$B_L \in [21, 25]$	$AM \leftarrow B$ and $MS \leftarrow Y$	
$\mathbf{C_{16}}$:Z_A-$B_L \in [30, 35]$	$AM \leftarrow B$ and $MS \leftarrow Z$	
$C_{17}(R)$:E-$S \in [0, 180]$		

Table 2.5: Episodes in Rich's mission TPNU

Similar to the earlier example, without any relaxations, there is no solution that

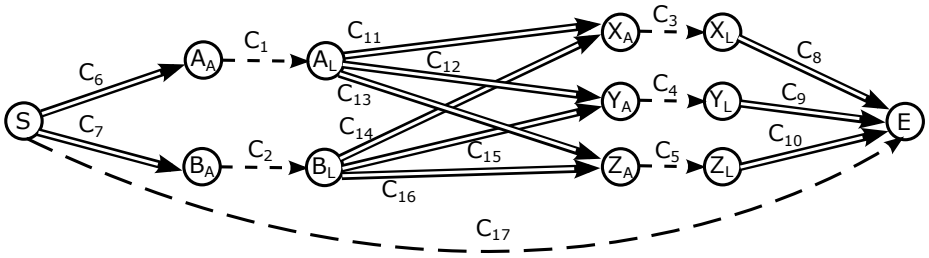


Figure 2-7: A graphical representation of Rich's mission TPNU

can satisfy all of the requirements in Rich's mission. If we use the TPN formalism to model this over-constrained problem, one solution would be to visit site B and X while extending the mission by 5 minutes (Figure 2-8). However, it does not account for the uncontrollable traversal times: the solution is very likely to fail during the mission, since it has no margin to absorb any delay in the traversal between locations. Next, we present two solutions generated for the TPNU model, which are based on two execution strategies that take the uncertainty into consideration. The first strategy, called *Strong Controllability*, comes up with a schedule of activities before starting the plan, which ensures success for all uncontrollable durations. The second execution strategy, called *Dynamic Controllability*, instead observes these uncertain outcomes along the way, and makes *more informed* decisions about scheduling each activity.

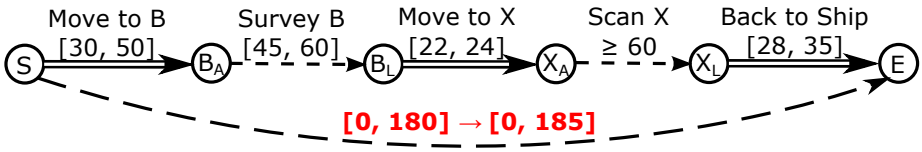


Figure 2-8: Consistent relaxation for Rich's mission

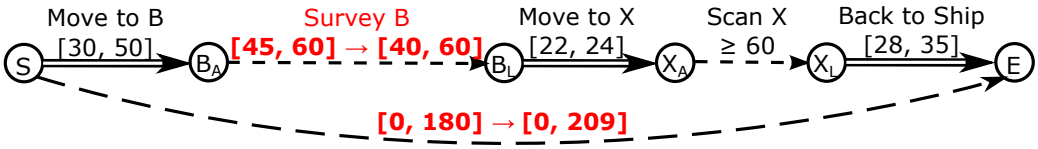


Figure 2-9: Strongly controllable relaxation for Rich's mission

The second solution is computed based on strong controllability (Figure 2-9). It extends the mission time to 209 minutes, and decreases the lower bound of the survey time at B to account for the uncertainty in the traversal between the ship and site

B. This solution has a utility of 83.9, and enables a schedule that satisfies Rich's requirements while being robust to the uncertain durations.

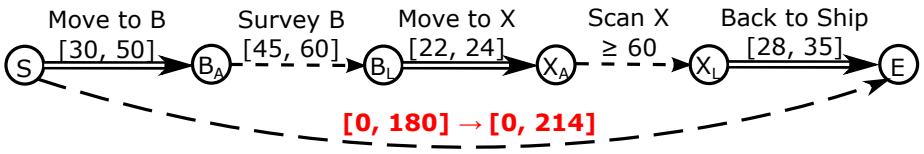


Figure 2-10: Dynamically controllable relaxation for Rich's mission

The third and final solution is computed based on dynamic controllability (Figure 2-10). Unlike the strong controllability solution, it does not need to decrease the lower bound of survey time at B to account for the uncertain traversal times, and hence is less conservative than the second solution, while still being safe. The solution has a higher utility of 57.4, and enables a dynamic schedule on the fly instead of a static schedule beforehand: the times of leaving site B and X will depend on the actual traversal times.

Definitions

Simple Temporal Networks with Uncertainty (STNUs) (Vidal & Fargier, 1999) have been widely used to model temporal problems with uncertain durations. They are extension to Simple Temporal Networks (Dechter et al., 1991) by adding a new class of constraint: *uncertain duration*. The duration is defined by a random variable between its lower and upper bounds and cannot be freely assigned. Formally, this structure is defined as the following:

Definition 11. *A STNU $N = \langle V_a, V_r, E_f, E_u \rangle$, extends the STN definition with new types of events and constraints associated with the uncertain duration:*

- *activated events $v_i \in V_a$, whose times are assigned by the agent;*
- *received events $r_i \in V_r$, whose times are assigned by external world;*
- *free constraints $e_i \in E_f$, which are of type $v_x - v_y \in [l_{xy}, b_{xy}]$ where v_x, v_y are events.*

- *uncertain durations* $u_{xy} \in E_u$*, where* $u_{xy} \in [l_{xy}, b_{xy}]$ *describes the difference in time between received event* $r_y \in V_r$ *and activated event* $v_x \in V_a$*, such that* $(r_y - v_x) = u_{xy}$*. Uncertain durations are not assigned by the agent, and can take any value in the bounded interval.*

The STNU extends the STN representation with uncertainty. Note that the uncertainty representation is not probabilistic: the uncertain durations are not associated with probability distributions. Thus, we can not reason over the likelihood of outcomes for the uncertain durations. We must instead guarantee constraint satisfaction given any outcome for the uncertain duration within the interval.

The solution to a STNU is an execution strategy that satisfies all constraints regardless of the outcomes of uncertain duration. The existence of such a strategy is characterized by the controllability, instead of consistency, of the STNU. There are three types of controllability (Vidal & Fargier, 1999): *Strong*, *Dynamic*, and *Weak*. Each type has a different assumption about the time when the outcomes of uncertain duration become available. In this thesis, we focus on the first two types, strong and dynamic controllability, which assume that no outcome is known prior to the execution. Strong controllability requires a predetermined schedule which satisfies all constraints regardless of the outcomes of the uncertain durations, whereas dynamic controllability requires a policy for scheduling as observations of uncertain durations become available. Intuitively, dynamic controllability is more flexible as it makes use of information gained during execution.

Definition 12. *A Simple Temporal Problem with Uncertainty, STPU, is a problem formulation using the STNU representation.*

- *Given a STNU* $N = \langle V_a, V_r, E_f, E_u \rangle$*, find an execution policy to events in* V_a *such that all constraints in* E_f *are satisfied regardless of the outcomes of durations in* E_u*.*

The STPU formalism has been extended with disjunctions and conditional constraints to handle more real-world scheduling and planning problems. In (Venable &

Yorke-Smith, 2005; Peintner, Venable, & Yorke-Smith, 2007), the *Disjunctive Temporal Problem with Uncertainty* (DTPU) formalism was introduced to permit non-convex and non-binary constraints. DTPU can be viewed as an extension to the deterministic Disjunctive Temporal Problem (DTP) (Stergiou & Koubarakis, 1998) formalism with uncertain durations. It allows the expression of disjunctive constraints, and enables the agent to choose between alternatives. We extends TPN to TPNU in a similar manner. Formally, the TPNU formulation extends the TPN definition with three additional elements.

Definition 13. *A TPNU contains all elements in a TPN, plus V_r, E_u and RE_u, where:*

- *$V_r \subseteq V$ is a set of received events. $V \backslash V_r$ is the set of all activated events;*
- *$E_u \subseteq E$ is a set of episodes with uncertain duration between pairs of activated and received events. $E \backslash E_u$ is the set of all episodes with free constraints;*
- *$RE_u \subseteq E_u$ is a set of continuously relaxable episodes with uncertain durations whose bounds can be tightened, and $RE_u \subseteq RE$.*

The cost function is generalized to include episodes with both controllable and uncertain duration: $f_e : (e_i, e_i') \rightarrow \mathcal{R}^+$ is a function that maps the following to a ***non-negative cost****.*

- *the* ***relaxation*** *of an episode with controllable duration, $e_i \rightarrow e_i', e_i \in RE \backslash RE_u$;*
- *or the* ***tightening*** *of an episode with uncertain duration, $e_i \rightarrow e_i', e_i \in RE_u$;*

As can be seen in the definition, we also generalize the concept of relaxations to include uncertain durations. To resolve a conflict by relaxing episodes with controllable temporal bounds, we will either increase its upper bound or reduce its lower bound, effectively widening the range of the temporal constraint. On the other hand, we shrink the bounds of uncertainty we handle for episodes with uncertain duration: we may resolve the conflict by increasing the lower bound and/or decreasing the upper

bound of its uncertain duration, effectively reducing the amount of uncertain outcomes to be handled. Usually, uncertain duration is used to reserve some flexibility for the agents or the environment in executing their activities. A tighter duration means less flexibility for them, but also imposes less restriction on the solution to the temporal plans. We will give more insights into the relation between uncertain durations and conflict resolutions in the approach section.

Similar to TPN, the solution to a continuously relaxable TPNU is defined as a 5-tuple $\langle A, S, R_e, R_u, E' \rangle$, where:

- A is a complete set of assignments to variables in P;
- S is a set of additional assignments that defines a total ordering over activities in the TPN;
- R_e is a set of continuous temporal relaxations for some episodes with controllable durations in RE.
- R_u is a set of continuous temporal tightenings for some episodes with uncertain durations in RE_u.
- E' is a set of episodes that encodes the traversal activities between activities, generated by the routing function. Each $e' \in E'$ encodes the traversal time associated with an agent's movement between locations specified in A, following the order defined by S.

A feasible solution to a TPNU provides a grounded and controllable STPU. We separate the solutions into two categories: *strongly controllable* and *dynamically controllable*. This is based on the type of execution strategies a solution can enable.

- A strongly controllable solution makes the resulting STPU strongly controllable. That is, the relaxation enables an execution strategy with a firm schedule for all events.

- A dynamically controllable solution makes the resulting STPU dynamically controllable, for which a dynamic execution strategy that meets all constraints can be derived.

Note that a strongly controllable solution is also a dynamically controllable solution, since strong controllability is more restrictive than dynamic controllability. Due to the flexibility of dynamic controllability, there is usually a greater solution space to explore for over-subscribed temporal plans.

2.4.2 Plans with Probabilistic Uncertain Durations and Chance Constraints

In many situations, modeling the uncertainty in temporal duration with a set-bounded representation is overly conservative, resulting in a loss of schedule utility. Chance-constrained formalisms, such as chance-constrained probabilistic Simple Temporal Problems (cc-pSTPs), address over-conservatism by imposing bounds on risk, while maximizing utility subject to these risk bounds. On the other hand, when we are dealing with probabilistic uncertain duration, the relaxation problem becomes more challenging, since we are making trade-offs between not only temporal requirements, but also risk taken. In the rest of this section, we present an extended formalism to TPN, the chance-constrained probabilistic Temporal Plan Networks (cc-pTPNs) for modeling relaxation for plans with probabilistic durations and chance constraints.

We again use Rich's mission as an example to illustrate the extension. To simplify the example, we remove the asphalt mound site survey, leave only the methane seeps sites to visit, and make the traversal duration controllable. The periodic methane seeps at X is likely to occur at around 1:00PM, following a normal distribution with a standard deviation of 30 minutes. The seeps at Y will occur at around 1:30PM and follows a normal distribution with a standard deviation of 50 minutes. As described before, Sentry leaves the ship at 11:00AM, and needs to arrive at the site before the start of the methane seeps. In addition, at least 30 minutes is required for traversing to the site, and 45 minutes for scanning. As before, Rich wants the mission to complete

in 3 hours, with less than 5% risk of violating **any** temporal requirements, such as returning late or missing the event. We can capture this problem using a cc-pTPN (Figure 2-11).

Table 2.6: Episodes in Rich's mission cc-pTPN

C_1	X_A-$S > [45, +\infty]$	$MS \leftarrow X$	Traversal to methane seeps site X
C_2	X_{sp}-$X_A \in [0, +\infty]$	$MS \leftarrow X$	Wait for seeps at site X
C_3(R)	X_L-$X_{sp} \in [50, 60]$	$MS \leftarrow X$	Scanning at site X
C_4	E-$X_L \in [45, +\infty]$	$MS \leftarrow X$	Traversal from site X back to ship
$\mathbf{C_5}$	X_{sp}-S=$[\mu = 120, \sigma = 30]$	$MS \leftarrow X$	Seeps start time at X
C_6	B_A-$S \in [30, +\infty]$	$MS \leftarrow Y$	Traversal to methane seeps site Y
C_7	Y_{sp}-$Y_A \in [0, +\infty]$	$MS \leftarrow Y$	Wait for seeps at site Y
C_8(R)	Y_L-$Y_{sp} \in [45, 60]$	$MS \leftarrow Y$	Scanning at site Y
C_9	E-$Y_L \in [30, +\infty]$	$MS \leftarrow Y$	Traversal from site Y back to ship
$\mathbf{C_{10}}$	Y_{sp}-S=$[\mu = 150, \sigma = 50]$	$MS \leftarrow Y$	Seeps start time at Y
C_{11}(R)	E-$S \in [0, 180]$		Mission duration

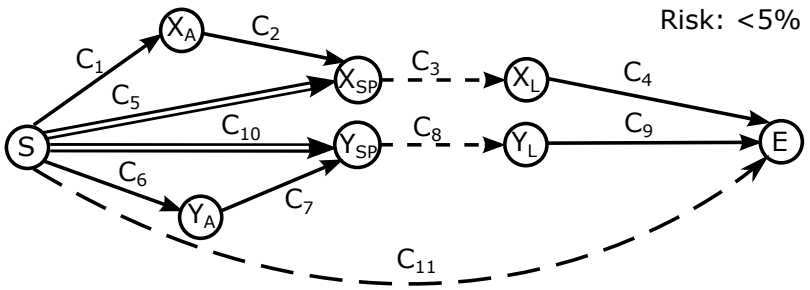


Figure 2-11: The cc-pTPN model for Rich's mission

After evaluating all the requirements, BCDR determines that no dynamic execution policy exists that meets all requirements. It engages Rich and starts presenting relaxations that can restore the feasibility of Rich's problem. The first one asks Rich to **extend the mission** from 3 hours to 4 hours and 26 minutes, which is robust for surveying the methane seeps at site X if it occurs between 11:45AM and 1:51PM (Figure 2-12). The probability of failure is determined by analyzing the assumptions on the uncertain duration: the episode C_5 with probabilistic duration is turned into a set bounded one with bounds [45,171], and the network is checked to be dynamically controllable against all possible outcomes within the range.

However, Rich rejects the solution and adds an additional requirement that the

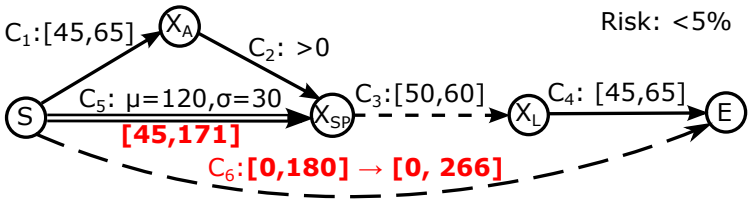


Figure 2-12: First relaxation for Rich's mission

mission duration can be at most 4 hours, since the subsequent mission cannot be shortened by more than an hour. BCDR incorporates this new requirement and generates another relaxation, which requires Rich to accept an increased probability of failure, from 5% to 20.85%. This allows the mission to be completed in 4 hours, but cannot account for methane seeps that occurs after 1:25PM (Figure 2-13).

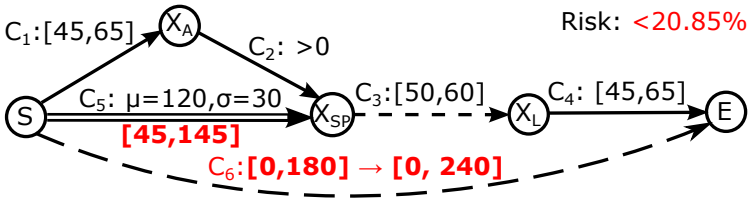


Figure 2-13: Second relaxation for Rich's mission

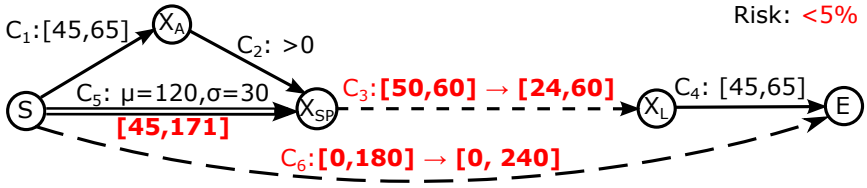


Figure 2-14: Third relaxation for Rich's mission

Rich rejects the solution again and tells BCDR that he cannot take more than 5% risk. BCDR again incorporates this new requirement and generates the third relaxation, which asks Rich to reduce the survey time at site X to 24 minutes. This allows the mission to be completed in 4 hours, while being robust to methane seeps that takes place between 11:45AM and 1:51PM (Figure 2-14).

Definitions

Similar to the STNU formalism, cc-pSTP is also an extension to the Simple Temporal Problems (STPs) formulation. In addition to the simple temporal constraints in STPs, it adds two new types of constraints to the problem: probabilistic temporal constraints for modeling uncertain durations, and chance constraints for specifying the acceptable level of risk. Compared to the set-bounded uncertain durations used in STNUs, the probabilistic representation of uncertain durations allows cc-pSTPs to more accurately model uncertainty in real world activities. In addition, the chance constraint supports a quantified bound on risk taken to be specified, which is more flexible and intuitive than the criteria of controllability. It can be viewed as a generalization of the notion of controllability: instead of a binary outcome between 100% guarantee of success or nothing, the users can ask for any bound between [0,100%] on the probability that a temporal network is executable.

Chance-constraints have been studied in the operations research literature, traditionally as probabilistic guarantees over satisfaction of individual or conjunctions of constraints (Kall, 1976). In this work, we generalize the definition of chance-constraints to include disjunctions over constraints as well. The definition is presented in Appendix A. This generalization allows us to choose between multiple options to find one which meets our safety requirements. We define the cc-pTPN formalism by extending TPNU with probabilistic uncertain durations and chance constraints. Here, we first repeat the definition of cc-pSTP from (Fang et al., 2014) for reference.

Definition 14. *A cc-pSTP is a pair* $\langle N^+, \Delta_t \rangle$*, where:*

- N^+ *is a probabilistic Simple Temporal Network (pSTN), defined as a 4-tuple* $\langle V_a, V_r, E_f, E_d \rangle$*, where:*
  - V_a, V_r, E_f *are defined as for STNU; and*
  - E_d *is a set of probabilistic uncertain durations. Each* $d_{xy} \in E_d, d_{xy} : \Omega \rightarrow \mathbb{R}$ *is a random variable describing the difference* $(y - x)$ *between an activated event* $x \in V_a$ *and a received event* $y \in V_r$*.*

- $\langle E_f, 1 - \Delta_t \rangle$ *is the chance constraint that sets the upper bound on the risk of failure, for the set of requirement constraints* E_f *in* N^+.

In the cc-pSTP, the set of constraints is described by E_f, which are difference constraints between elements in V_r and V_a to be satisfied, given the outcomes of E_d. This corresponds to C in Definition 23.

Definition 15. *A cc-pSTP solution is a pair* $\langle N_g, S_x \rangle$*, where:*

- N_g *is a grounded STNU of the cc-pSTP. It replaces all probabilistic durations in the cc-pSTP with set-bounded uncertain durations, which specify the allocated risk over them: the lower and upper bounds allocated for each probabilistic duration indicate the range of outcomes covered. The total probability of uncovered outcomes across all probabilistic durations must be smaller than the chance constraint bound.*

- S_x *is an execution strategy for* N_g*. It covers all controllable events in the cc-pSTP, and is controllable with respect to* N_g*.*

Given a cc-pSTP, P, if we execute the controllable events using S_x in its solution, the chance of violating any temporal constraints in P is guaranteed to be less than Δ_t. The policy S_x could be a static schedule (with a strongly controllable N_g), or a dynamic execution policy (with a dynamically controllable N_g). If a cc-pSTP is over-constrained, no solution exists that can meet all temporal constraints within the risk bound. In other words, there is no N_g that is controllable and takes less risk than the chance constraint. This occurs when the user specifications are too restrictive, for example when the desired time bounds are too tight, or when the user is overly cautious in setting the chance constraint. These problems can be resolved through temporal or chance constraint relaxations: they are trade-offs between weakening over chance and temporal constraints for the users. We thus define a relaxable version of the cc-pSTP formulation, which allows some of its constraints to be relaxed at a cost to allow a feasible solution.

Definition 16. *A relaxable cc-pSTP contains all elements from a cc-pSTP plus four additional elements, RE, f_{rc}, $r\Delta_t$ and f_Δ, where:*

- *RE is a set of requirement constraints whose temporal bounds can be relaxed, $RE \subseteq E_f$.*
- *$f_e : (e_{ij}, e'_{ij}) \rightarrow \mathbb{R}^+$ is a function that maps the relaxation of a relaxable constraint, $e_{ij} \rightarrow e'_{ij}$ where $e_{ij} \in RE$, to a positive cost value.*
- *$r\Delta_t \in [T, F]$ is a boolean value that indicates if the chance constraint can be relaxed.*
- *$f_\Delta : (\Delta_t, \Delta'_t) \rightarrow \mathbb{R}^+$ is a function that maps the relaxation of the chance constraint, $\Delta_t \rightarrow \Delta'_t$ where $\Delta_t \leq \Delta'_t \leq 1$, to a positive cost value.*

Definition 17. *A valid resolution for an over-constrained cc-pSTP, P, is a 3-tuple $\langle R_e, \Delta'_t, N_{alloc} \rangle$, where:*

- *R_e is a set of relaxations (in terms of relaxed lower and upper bounds) to constraints in RE of P.*
- *Δ'_t is a relaxation for Δ_t of P, and $\Delta'_t \geq \Delta_t$.*
- *N_{alloc} is a STNU generated from P by grounding all probabilistic durations with fixed lower and upper bounds.*

such that N_{alloc} is controllable and covers more than $1 - \Delta'_t$ of the uncertain durations' outcomes.

Based on the relaxation problems for cc-pSTP, we define a chance-constrained analogue to the TPNU, the chance-constrained probabilistic Temporal Plan Network (cc-pTPN).

Definition 18. *A cc-pTPN contains all elements in a TPNU (Definition 13), with the following differences:*

- *Instead of $E_u \subseteq E$, a set of episodes with uncertain duration between pairs of activated and received events in TPNU, the cc-pTPN features a set of episodes with probabilistic uncertain durations E_d. Each $d_{xy} \in E_d$ is a random variable describing the difference $(y-x)$ between an activated event and a received event, where $x \in V_a$ and $y \in V_r$;*

- *A cc-pTPN includes $\langle E_f, 1-\Delta_t \rangle$, the chance constraint that sets the upper bound on the risk of failure, for the set of requirement constraints $E_f \subseteq E$;*

- *A cc-pTPN includes a boolean value $r\Delta_t \in [T, F]$ that indicates if the chance constraint can be relaxed;*

- *The cost function is adapted for the chance constraint with the addition of $f_\Delta : (\Delta_t, \Delta_t') \rightarrow \mathbb{R}^+$, a function that maps the relaxation to the chance constraint, $\Delta_t \rightarrow \Delta_t'$ where $\Delta_t \leq \Delta_t' \leq 1$, to a positive cost value.*

Intuitively, the cc-pTPN formulation adds probabilistic uncertain durations with an associated chance-constraint, in addition to the set-bounded uncertain durations in TPNUs. Correspondingly, the relaxation over the risk bound is encoded by the tightness of the chance constraint.

The concept of relaxing an uncertain duration is related to the concept of relaxing a chance constraint. In finding a solution to cc-pTPN, we are required to find an example set of episodes with set-bounded uncertain durations which cover enough probability mass to satisfy the chance constraint. A relaxation of the chance constraint allows the choice of episodes with set-bounded uncertain durations to cover a smaller probability mass, in some cases a more restrictive set of outcomes. This is analogous to the relaxation of the uncertain durations in the original TPNUs.

Similar to TPNU, the solution to a continuously relaxable cc-pTPN is defined as a 6-tuple $\langle A, S, R_e, \Delta_t', N_{alloc}, E' \rangle$, where:

- A is a complete set of assignments to variables in P;

- S is a set of additional assignments that defines a total ordering over activities in the TPN;

- R_e is a set of continuous temporal relaxations for some episodes with free constraints in RE.
- Δ'_t is a relaxation for Δ_t, and $\Delta'_t \geq \Delta_t$.
- N_{alloc} is a TPNU that grounds all episodes with probabilistic durations using fixed lower and upper bounds.
- E' is a set of episodes that encodes the traversal activities between locations, generated by the routing function. Each $e' \in E'$ encodes the traversal time associated with an agent's movement between locations specified in A, following the order defined by S.

A feasible solution to a cc-pTPN provides a grounded and controllable STPU. We can also separate the solutions into two categories, *strongly controllable* and *dynamically controllable*, based on the type of execution strategies a solution can enable.

Given an over-subscribed cc-pTPN, there is usually more than one valid resolution to it due to the continuous property of temporal and chance constraint relaxations. It is important to prioritize the resolutions and enumerate only preferred ones of lower cost for the users. In addition, finding a good resolution usually requires a considerable amount of negotiation since the users may not have encoded all their requirements in the input problem. BCDR needs to learn about them through the interaction before reaching an agreement with the user.

2.5 Domain Relaxations

2.5.1 Example Scenario

For the fourth and final type of relaxation, domain relaxation, we switch back to use the urban travel example to illustrate the extensions required to the TPN and its solution formalisms. Recall that Uhura's task is to work out a temporally feasible plan with Simon and Christian for their evening outing trip, which includes the choices for restaurant and movie showing, the sequence of these activities, and appropriate

adjustments to the timing requirements, if necessary. We extend the variable representation for their activities to be parameterized, which include domain constraints that encode the meanings as sets of semantic queries. These semantic queries capture the user requirements on the state that cannot be encoded by only using temporal constraints, and describe allowed values for the domain of these variables.

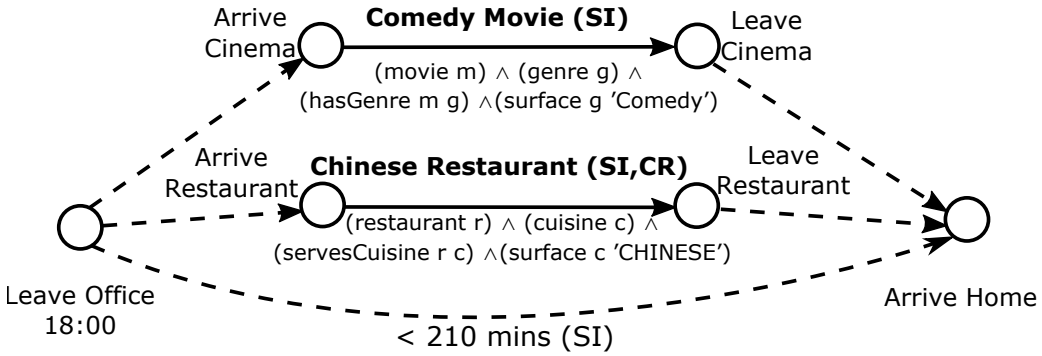


Figure 2-15: A TPN for Simon's trip

For example, the extended TPN for Simon and Christian's travel problem is shown in (Figure 2-15), which again encodes the two activities for dinner and movie, as well as the temporal requirements over the trip departure and completion times. The two activities requested by them are associated with a set of semantic queries that encode the genre and cuisine requirements. When implemented with SparQL, these queries can be expressed as the following (m.05p553 and m.01xw9 are Freebase Machine IDs for entity *Comedy film* and *Chinese food*):

- **Comedy Movie**:

  SELECT ?m WHERE{ *//select subject ?m that meets the following triples*

  ?m ns:type.object.type ns:film.film. *//subject ?m is of type film*

  ?g ns:type.object.type ns:film.film_genre. *//subject ?g is of type genre*

  ?m ns:film.film.genre ?g. *//?m has genre ?g*

  FILTER (?g = <http://rdf.freebase.com/ns/m.05p553>).} *//?g is object m.05p553*

- **Chinese Restaurant**:

  SELECT ?r WHERE{

  ?r ns:type.object.type ns:dining.restaurant.

  ?c ns:type.object.type ns:dining.cuisine.

  ?r ns:dining.restaurant.cuisine ?c.

  FILTER (?c = <http://rdf.freebase.com/ns/m.01xw9>).}

Given the domain constraints of the variables for dinner and movie, we pass their semantic queries to a knowledge base, which can search through multiple data sources and retrieve candidate options for the activities. These options will then be encoded as alternative episodes for the activities and added to the TPN. For example, the expanded TPN for Simon and Christian's outing trip is shown in Figure 2-16.

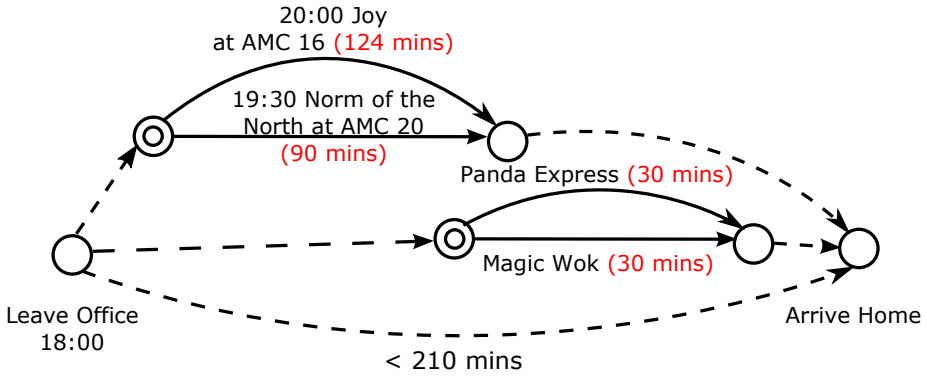


Figure 2-16: An expanded TPN with alternative activity candidates

Due to the long travel times to and from the candidate Chinese restaurants, no solution can be found that meets all temporal requirements. Hence the system engages Simon and Christian, and initiates a discussion about possible resolutions for their

problem.

> **DA**: *Simon, you may have dinner at Magic Wok then watch the 8pm Joy at AMC 16. However, due to the length of the movie you won't be back home until 10:34pm. Is that OK?*
>
> **Simon**: *No, I must arrive home by 9:30pm.*
>
> **DA**: *OK, then Simon can you leave office 30 minutes earlier? If so you may watch Norm of the North at 7:30pm, and arrive home at 9:30pm.*
>
> **Simon**: *No I cannot leave office before 6pm.*
>
> **DA**: *Simon and Christian,* ***How about eating at Sunny Bowl, a Korean restaurant?*** *It is closer and Simon can make the 7:30pm movie without leaving any earlier.*
>
> **Christian**: *That's fine.*
>
> **Simon**: *Sounds good. Thank you.*

In this example, Simon cannot change the departure and arrival times. As a result, he rejected the first two proposals. Previous approaches would have failed at this step, as no more temporal relaxation can be found that resolves the conflicts between long travel times to the restaurants and movie start times. However, the domain relaxation extension to BCDR weakens the domain constraints for the restaurant variable, such that three new options became available for his trip (Figure 2-17). In this case, BCDR discovered a close alternative, Korean, for the cuisine requirement of restaurant. It then queried the knowledge base to retrieve additional candidate restaurants, and found one that is closer to their home and satisfies all temporal constraints (Figure 2-18).

This example demonstrates the desired behavior of domain relaxation: it allows BCDR to resolve over-subscribed travel plans through relaxing the domain constraints, and actively searching for candidates that are not encoded in the original problem. It gives the users more flexibility in resolving their over-subscribed plans when they cannot compromise on the temporal requirements.

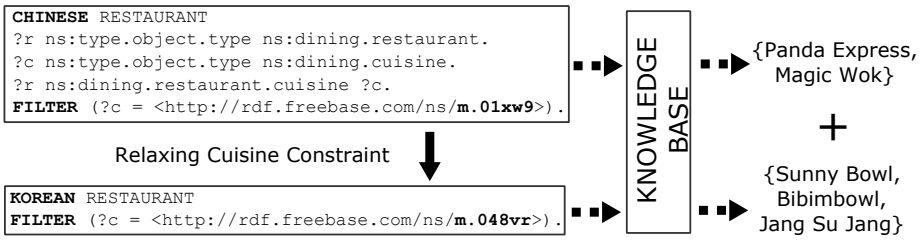
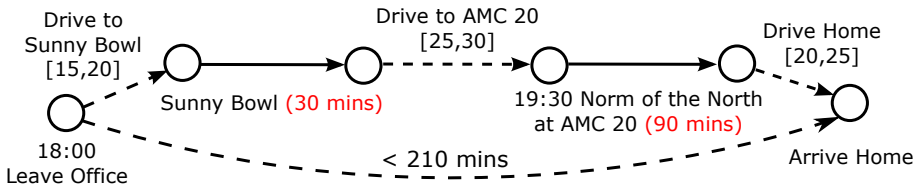
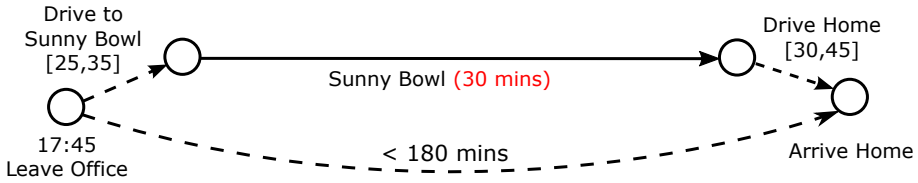


Figure 2-17: Domain relaxation for the restaurant cuisine



(a) Simon's trip



(b) Christian's trip

Figure 2-18: A solution enabled by relaxed cuisine constraint

2.5.2 Definitions

To support domain relaxation, we augment the TPN formulation to include relaxable domain constraints for the variables. Formally, we define a relaxable Temporal Plan Network with domain constraints, such as the one presented in Figure 2-16, as the following:

Definition 19. *A relaxable TPN with domain constraints is an 11-tuple* $\langle P, Q, V, E, RE, L_e, L_p, f_p, f_e, P_s, L_s \rangle$ *where:*

- P *is a set of controllable finite domain discrete variables;*
- Q *is the collection of domain assignments to* P*;*
- V *is a set of events representing designated time points;*
- E *is a set of episodes between pairs of events* $v_i \in V$*;*

- *$RE \subseteq E$ is a set of relaxable episodes whose temporal bounds can be continuously relaxed;*

- *$L_e : E \rightarrow 2^Q$ is a guard function that attaches conjunctions of assignments in Q, $q_i \in Q$, to some episodes $e_i \in E$;*

- *$L_p : P \rightarrow 2^Q$ is a guard function that attaches conjunctions of assignments in Q, $q_i \in Q$, to some discrete variable $p_i \in P$;*

- *$f_p : Q \rightarrow \mathcal{R}^+$ is a function that maps each assignment to every controllable discrete variable, $q_i \in Q$, to a positive **reward** value;*

- *$f_e : (e_i, e_i') \rightarrow \mathcal{R}^+$ is a function that maps the relaxation to one relaxable temporal constraint $e_i \in RE$, from e_i to e_i', to a positive **cost** value.*

- *P_s is a set of domain constraints, where each $P_{si} \in P_s$ is a semantic query;*

- *$L_s : P \rightarrow S$ is a function that attaches semantic constraints, $s_i \in S$, to some variables $p_i \in P$, which defines the domain of the variable.*

As presented in the previous section, some of the variables in the TPN are associated with domain constraints to encode their meanings, such as the example presented in Figure 2-16. They are highlighted in the definition: domain constraints P_s and function L_s that associates the constraints to the corresponding variables. These domain constraints, represented by semantic queries, are used to retrieve domain values (encoded as object bindings) from the knowledge base. In the case of over-subscription, some of the queries can be relaxed, which effectively weakens the domain constraints for a variable and allows additional options in the domain for resolving conflicts. In this thesis, we use SparQL as the implementation for the semantic queries. And formally, we encode the SparQL query as a 4-tuple $\langle N, H, W, RW \rangle$, where:

- N is the namespace of the query;

- H is the SELECT clause, which identifies the variables to appear in the query results;

- W is a collection of SparQL query triples, each contains a subject, predicate and object field. They are encoded as part of the WHERE clause;
- $RW \subseteq W$ is a set of relaxable triples, whose object field can be modified to other values.

For example, Figure 2-17 demonstrates the relaxation to a domain constraint for the *Restaurant* variable. The original set of triples in the SparQL query retrieves only two Chinese restaurants, and neither of which meets Simon's tight temporal requirements. A domain relaxation for this variable weakens the FILTER triple, allowing its object to be Korean cuisine in addition to Chinese cuisine, which adds three more restaurants to be considered. Internally, all objects are encoded using their unique Freebase Machine IDs (MIDs), such as m.01xw9 for Chinese cuisine and m.048vr for Korean cuisine, to avoid ambiguity.

Note that it is also possible to include uncertain temporal durations and chance constraints in the TPN using the extensions for TPNUs and cc-pTPNs. They are necessary for generating risk-bounded relaxations for many real-world problems. As presented in Section 2.4, BCDR is capable of checking controllability and computing risk-bounded relaxations to the uncertain durations. For simplicity, we omit the uncertain durations from our extended formulation for domain relaxation in this section.

With the extensions for temporal and domain relaxations, the output of BCDR is now a 5-tuple, which defines a temporally feasible plan based on the input TPN. It may also include continuous relaxations for some temporal constraints, and domain relaxations for some variables, if necessary.

Definition 20. *The solution to a relaxable TPN with domain constraints is a 5-tuple* $\langle A, S, R_e, R_d, E' \rangle$*, where:*

- A *is a complete set of assignments to variables in* P*;*
- S *is a set of additional assignments that defines a total ordering over activities in the TPN;*
- R_e *is a set of continuous temporal relaxations for some episodes in* RE*.*

- *R_d is a set of domain relaxations for some variables in P;*
- *E' is a set of episodes that encodes the traversal activities between locations, generated by the routing function. Each $e' \in E'$ encodes the traversal time associated with an agent's movement between locations specified in A, following the order defined by S.*

For example, in the evening outing trip example, BCDR relaxes the cuisine constraint of the restaurant from Chinese to Korean in order to find a feasible solution. In addition to the new domain values, domain relaxation may also introduces new episodes into the problem, such as the travel times to and from the new restaurant. In this thesis, we discuss the application of BCDR to travel problems with location-tagged activities, since the temporal durations of these additional episodes in these scenarios are straightforward to compute. Generating episode for domain relaxations in some other domains can be very difficult, and it is not the focus of this thesis.

2.6 Chapter Summary

In this chapter, we presented the problem statement for resolving over-subscribed temporal plans. The input to the BCDR algorithm is a Temporal Plan Network that encodes all alternative plans and temporal requirements. The outputs are a grounded plan, and a set of discrete, continuous, risk-bounded and domain relaxations that is necessary to make it temporally feasible. In the following chapters, we will discuss the BCDR algorithm in details, and how it coordinates the different relaxation techniques to generate these relaxations for resolving conflicts in over-subscribed temporal plans.

Chapter 3

Conflict-Directed Relaxation for Temporal Plans

In this chapter, we present the Best-first Conflict-Directed Relaxation algorithm for detecting and resolving conflicts in over-subscribed temporal plans, and its application for generating discrete relaxations. Building upon prior work on diagnosis (de Kleer & Williams, 1987; Williams & Ragno, 2002), BCDR is capable of handling over-subscribed temporal plans with continuous variables and constraints. Instead of likely failure modes, it extracts conflicting choices and episodes to **explain the cause of failure** in a plan, and enumerates preferred resolutions in **best-first** order.

In recent literature, several approaches have been developed to solve over-constrained scheduling and planning problems, which are often framed as over-constrained CSPs (for over-constrained scheduling problems) and optimal planning problems (for over-subscribed planning problems). The major challenge in solving these problems is the enormous search space. In fact, the problem of finding all resolutions to an over-subscribed planning problem is NP-Complete (O'Sullivan, Papadopoulos, Faltings, & Pu, 2007), assuming that a polynomial algorithm exists that can check if a temporal plan is executable. On the other hand, in many of the over-subscribed scenarios, the users are often expecting a quick response and would like the algorithm to be very efficient in coming up with a resolution.

Many techniques, especially the techniques developed to solve CSPs, have been

implemented to speed up the search for resolutions, including standard and domain specific ones. For over-constrained scheduling problems, the techniques include forward checking, conflict-directed back jumping, Dualize & Advance (Bailey & Stuckey, 2005), removal of subsumed variables (Moffitt & Pollack, 2005b) and semantic branching (Armando, Castellini, & Giunchiglia, 1999). One other approach is to give up the requirement of completeness and use a local search algorithm, like (Beaumont et al., 2001). This approximate approach usually runs much faster than the systematic methods, however, it cannot guarantee the optimality or completeness of the results. For over-subscribed generative planning problems, they have often been framed as deterministic oversubscription planning (OSP) in literature (Domshlak & Mirkis, 2015). Optimal planning has been the primary approach for OSPs, in which the objective is reformulated as finding a plan to achieve a subset of the goals with higher rewards.

One additional issue with these over-subscribed problems are the large numbers of resolutions: facing thousands or even millions of resolutions, it is difficult for us humans to select the best one from them. This imposes a big challenge on resolving problems through simple and efficient communication. An effective approach must only present a few preferred ones to the user, and provide the rationale behind their relaxations, in order to let the user make an informed decision. In (O'Sullivan et al., 2007), an approach is presented to reduce the amount of resolutions by computing representative plan relaxations. It is based on the notion of *representative set*, in which all resolutions generated cannot be dominated by any other resolutions in the set.

We designed BCDR to take Temporal Plan Networks as input, and produce preferred relaxations that enable temporally feasible plans for each individual agent. This model is more general than the simple temporal problem formulations used in many prior works on temporal relaxation, and provides supports for multi-agent coordination and activity sequencing. In over-subscribed situations that no temporally feasible plans can be found, we would like to find preferred resolutions using alternatives for both activities and timing, and preserve as much flexibility as possible for the users. However, BCDR requires complete plans as input and focuses on restor-

ing temporal feasibility. This makes BCDR's relaxation problem strictly simpler than over-subscribed generative planning problems, which is left for future work to address.

Introduced in (Yu & Williams, 2013), BCDR was the first algorithm for learning hybrid conflicts in conditional temporal problems and enumerating preferred continuous temporal relaxations. It was later extended to handle uncertain durations (Yu et al., 2014), chance constraints (Yu et al., 2015), multi-agent vehicle routing problems (Yu et al., 2016b) and relaxable domain descriptions (Yu et al., 2016a). In this chapter we focus the basic configuration of BCDR that computes discrete relaxations. Section 3.1 describes the algorithm, and Section 3.2 presents how BCDR incorporates user feedbacks for improving solutions. The extensions for alternative conflict learning and relaxation generation techniques will be introduced in the following chapters.

3.1 Computing Discrete Relaxations for Over-subscribed Temporal Plans

In this section, we present the design and implementation of BCDR for resolving over-subscribed temporal plans. Given an abstract travel plan that encodes all requirements from the users, BCDR fills in the details by computing feasible sequences of activities, adding contingencies for likely delays during transit and generating alternatives for the temporal and destination requirements, if necessary. BCDR takes in a TPN, a set of agent models and a routing function as inputs, and produces a plan, as well as suspensions of some episodes if necessary.

BCDR leverages ideas from (Williams & Ragno, 2002) for efficient conflict detection and resolution, and generalizes methods from the Dualize & Advance algorithm (DAA, (Gunopulos, Khardon, Mannila, & Sharma, 2003)) for incrementally discovering conflicts and enumerating relaxations in best-first order. We first present an overview of BCDR, including its inputs, outputs, and key procedures; then discuss in details the two key features of the algorithm: computing relaxations for conflicts, and activity sequencing for each agent. Algorithm 1 presents the pseudo code of BCDR's

main function.

Input: A relaxable TPN $Tp = \langle P, Q, V, E, RE, L_e, L_p, f_p, f_e \rangle$, a set of agent models Ag, and a routing function f_r.
Output: A solution, $\langle A, S, R_e, E' \rangle$ that maximizes $\sum_i (f_{pi} - f_{ei})$.

```
1  Cand ← ⟨A, S, R_e, E', C_r⟩ // first candidate;
2  Seq ←GetSeqVariables(P) // the activity sequence variables;
3  Path ← new PATH(P, Seq) // path constraint over all activities;
4  Queue ← {Cand} // a priority queue of candidates;
5  C ← {} // the set of all known conflicts;
6  U ← P; //the list of unassigned controllable variables;
7  while Queue ≠ ∅ do
8      Cand ←Dequeue(Queue);
9      knownCFT ←UnresolvedConflicts(Cand, C);
10     if knownCFT == null then
11         if isComplete?(Cand, U) then
12             newCFT ←PropagatePATH(Cand, Path);
13             if newCFT == null then
14                 AddRoutes(Cand);
15                 newCFT ←TemporallyFeasible?(Cand);
16             endif
17             if newCFT == null then
18                 return Cand;
19             else
20                 C ← C ∪ {newCFT};
21                 Queue ← Queue ∪ {Cand};
22             endif
23         else
24             Queue ← Queue∪ExpandOnVariable(Cand, U);
25         endif
26     else
27         Queue ← Queue∪ ExpandOnConflict(Cand, knownCFT);
28     endif
29 end
30 return null;
```

Algorithm 1: The BCDR algorithm for computing discrete relaxations

The Conflict-Directed A* (CD-A*) enumerates the best solutions to finite-domain CSPs according to an objective function, and guides the search using conflicts learned from inconsistent sets of assignments. Once detected, a conflict is used to prune the search space by extending each partial candidate with alternative resolutions. Like CD-A*, BCDR takes an A* search strategy by evaluating each partial candidate

using an admissible heuristic function and expanding the search tree in best-first order. Hence the first relaxation found is guaranteed to be the best one.

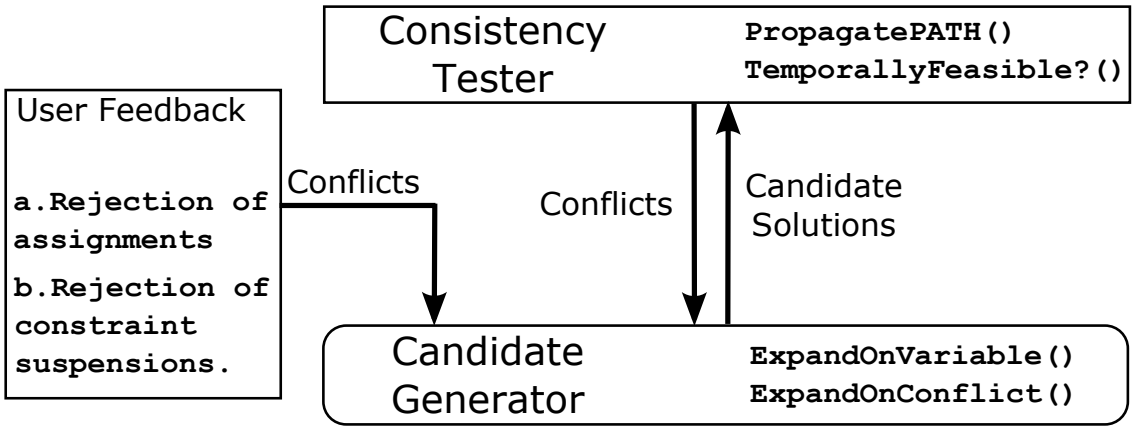


Figure 3-1: The generate and test architecture used by BCDR

As presented in Chapter 1, the generate-and-test approach includes two key components: consistency tester and candidate generator (Figure 3-1). The tester is implemented as Function PROPAGATEPATH (Line 12) and TEMPORALLYFEASIBLE? (Line 15), which ensure that the candidates enable consistent plans that meet the temporal requirement. If not, they learn conflicting constraints from the over-subscribed plans and feed them to the candidate generator. The generator is implemented as Function EXPANDONVARIABLE (Line 24) and EXPANDONCONFLICT (Line 27), which explore the search space and generate new candidate solutions using two types of expansions: *Expand on an unassigned variable* and *Expand on an unresolved conflict*. The first expansion guides the search into unexplored regions, and the second expansion keeps the search away from known infeasible regions in the search space.

BCDR starts with an empty candidate in the queue (Line 1). A candidate, *Cand*, is a 5-tuple $\langle A, S, R_e, E', C_r \rangle$ with goal assignments A, sequential assignments S, constraint suspensions R_e, additional episodes E' and resolved conflicts C_r, all being empty lists in the first candidate. BCDR continues looping until the first candidate is found that makes the input problem consistent (Line 17). If BCDR does not find a consistent candidate and the queue is exhausted, it returns null indicating that no relaxation exists for the input problem (Line 30).

Within each loop, BCDR first dequeues the best partial candidate, *Cand* (Line 8). It checks if *Cand* resolves all known conflicts (Line 9). If not, an unresolved conflict *knownCFT* will be returned by function UNRESOLVEDCONFLICTS, which compares the resolved conflicts C_r in *Cand* with all known conflicts C. *knownCFT* is then used for expanding *Cand* by function EXPANDONCONFLICT (Line 27). The child candidates of *Cand* will then be queued. For example, assume that we need to expand a partial candidate {*RT*=PE} (*Panda Express for restaurant*) with conflict *MV*=NN (*Norm of the North for Movie*), BCDR will create two child candidates that extend the partial candidate using the alternative assignments of variable *MV*, JY (Joy), and suspension of the temporal constraint on arriving home (Figure 3-2b). The expanded candidates will be added back to *Queue*.

If *Cand* resolves all known conflicts, BCDR then checks if it is complete, which means that no more variables can be assigned, by comparing its assignments and all unassigned variables in the problem (Line 11). If *Cand* is incomplete, BCDR will expand it using the assignments to one unassigned variable through function EXPANDONVARIABLE (Line 24). For example, assume again that we need to expand a partial candidate {*RT*=PE}, but with variable *MV*:{*JV*,*NN*}. This time, we simply create two candidates that extends the partial candidate using the two possible assignments of *MV* (Figure 3-2a).

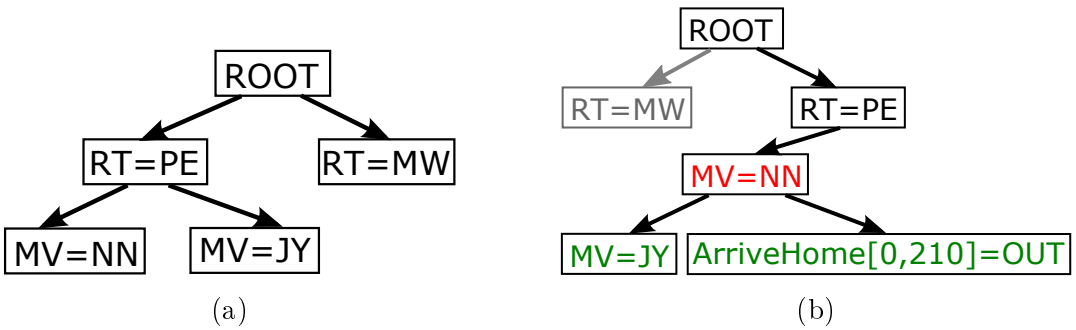


Figure 3-2: Examples of expanding on variable and conflict

If *Cand* is complete, BCDR proceeds to check its consistency using function TEMPORALLYFEASIBLE? (Line 15). If no conflict is detected, *Cand* will be returned as the best solution for the input problem (Line 18). If a new conflict, *newCFT*, is de-

tected by TemporallyFeasible?, BCDR will record it in C and put $Cand$ back to the queue for future expansions (Line 20,21), since it now has one unresolved conflict.

To support activity sequencing, we introduce a global constraint, $PATH$, into BCDR to ensure that each agent gets a consistent route from their origins to their destinations (Kilby & Shaw, 2006). $PATH$ is commonly used in modeling vehicle routing problems: it is one of the global constraints over discrete variables that ensures the vehicle visits all locations following a valid sequence. A valid path contains no break point in the middle, such that there is one and only one arrival and departure route to each intermediate waypoint. In addition, the first and final locations must be the origin and destination of the agent's trip. For BCDR's multi-agent vehicle routing problems, we define the $PATH$ constraint (Line 3) over the activity sequence variables (Line 2) for each agent, that is, the *'what to do next'* variable for each activity. In order to check candidate plans against $PATH$ constraint, we add an additional PropagatePATH function before the temporal feasibility checking function (Line 12). This function (Algorithm 2) is implemented based on the propagation techniques introduced in (Francis & Stuckey, 2014), which decomposes the $PATH$ constraint into two constraints in conjunctions: $AllDifferent$ and $NoSubtour$. The function checks their feasibility separately (Line 2 and 4), and returns a conflicting set of assignments if either of these checks fail. This approach is efficient in identifying invalid activity sequence, returning conflicting assignments and signaling BCDR to backtrack and try alternative orders. The addition of $PATH$ constraints and its propagation function allows BCDR to enumerate not only alternative options for activities, but also feasible ordering of them.

In addition, BCDR only computes a route and estimates the traversal time when necessary, instead of requiring all routes to be specified in the input problems. In BCDR, TemporallyFeasible? is the function that evaluates temporal feasibility for a candidate plan. Therefore, we add an additional function, AddRoutes (Algorithm 3), right before TemporallyFeasible?. At this step (Line 15 in Algorithm 1), BCDR has a candidate with a complete set of assignments to all activities, which is ensured by function IsComplete?, and a total ordering of them for each

Input: A candidate $C : \langle A, S, R_e, E', C_r \rangle$.
Output: A conflict over a subset of assignments in S, the activity sequencing assignments in C.

```
1 Conflict ← null // the initial conflict variable;
2 Conflict ←AllDifferent(S);
3 if Conflict == null then
4 |  Conflict ←CheckSubtour(S);
5 endif
6 return Conflict;
```

Algorithm 2: Propagate function for the PATH constraint

agent. The AddRoutes function iterates through each sequence assignment in the candidate, executes the routing function between the locations it connects, and creates a traversal episode that encodes the route and time. The routing function is implemented using an open source navigation package, GraphHopper (Graphhopper, 2015), with road data from OpenStreetMap (Haklay & Weber, 2008). Note that the basic configuration of BCDR uses a consistency model when checking temporal consistency, which only evaluates if a solution is consistent with one of the values within the bounds. In Chapter 5, We will discuss a few extensions for BCDR that generate more robust solutions subject to temporal uncertainty.

Input: A candidate $C : \langle A, S, R_e, E', C_r \rangle$, a routing function $f_r(O, D, m)$ and an agent i.

```
1 for s in S do
  |  // Compute route given the origin and destination specified by s,
  |  // and the mode of travel for the agent.
2 |  route ← f_r(Origin(a),Destination(s),m_i));
  |  // Create a travel activity using the route, and add to E'.
3 |  E' ← E' ∪ CreateEpisode(s, route);
4 end
```

Algorithm 3: BCDR's Routing function

3.2 Incorporating User Inputs

As demonstrated in the example, the user can add additional inputs given an unsatisfying solution, or requirements he/she forgot to encode in the original plan. BCDR will incorporate inputs into its search process and respect them in all future solutions.

Given a solution, two types of inputs can be accepted by BCDR while computing discrete relaxations:

- Rejection of an assignment, such as 'I do not want to visit Panda Express for dinner'.
- Rejection of the suspension of an episode, such as 'The arrival time requirement cannot be removed'.

BCDR utilizes the conflict-directed approach to efficiently adapt to these inputs. Instead of modifying the input problem and restarting the search process from the beginning, it will record the input as a new conflict and add it to the known conflicts list. The above two types of inputs will be recorded as following:

- Rejection of an assignment $X = a$: a new conflict $X = a$ will be created and added to BCDR's conflict collection, such that it will not appear again in any future solution.
- Rejection of a suspension of constraints. For relaxation $e_i = OUT$, a new conflict $e_i = OUT$ will be added to BCDR's conflict collection, such that all future attempt for suspending e_i will not be allowed.

The pseudo code of this implementation, called Reactive BCDR, is presented in Algorithm 4. Note that the BCDR algorithm presented earlier is wrapped inside the function BCDR. Reactive BCDR starts with querying BCDR for a solution to the given problem (Line 4). If no solution can be found to the problem, the algorithm will signal failure and terminate (Line 6). Otherwise, the first solution will be presented to the user. If the user accepts it, Reactive BCDR will return the solution and terminate (Line 9). If the user rejects it, it will prompt the user for additional inputs, and encode them into conflicts using the two rules (Line 12). Note that the current solution will also be put back to the queue, since it now has an unresolved conflict. If no input is provided, BCDR will move on to find the next best solution, and the current solution is discarded.

Input: A TPN Tp, a set of agent models Ag, and a routing function f_r.
Output: Sol: A valid relaxation, or *null* is none exists for the input problem.

```
1  C ← {} // the set of all known conflicts kept by BCDR;
2  Queue ← {} // the priority queue for candidate relaxations kept by BCDR;
3  while true do
4  |  ⟨Sol, C, Queue⟩ ←BCDR(Tp, Ag, f_r);
5  |  if Sol == null then
6  |  |  return null;
7  |  else
8  |  |  if ACCEPTED?(Sol) then
9  |  |  |  return Sol;
10 |  |  else
11 |  |  |  if UserInputs ≠ null then
12 |  |  |  |  C ← C∪PARSEINPUTS{UserInputs};
13 |  |  |  |  Queue ← Queue ∪ Sol
14 |  |  |  endif
15 |  |  endif
16 |  endif
17 end
```

Algorithm 4: REACTIVE BCDR for computing discrete relaxations

3.3 Chapter Summary

In this chapter, we presented the first contribution of the thesis: the Best-first Conflict-Directed Relaxation (BCDR) algorithm. BCDR is the basic framework for all relaxation techniques presented in this thesis, and in this chapter we present its application to computing discrete relaxations. Compared to prior work on generating relaxations for over-subscribed problems, BCDR is capable of incrementally discovering conflicts and enumerating relaxations in best-first order, while providing rationales for each relaxation generated based on the conflicts detected.

Chapter 4

Continuous Relaxation for Temporal Constraints

In this chapter, we present the continuous relaxation extension to BCDR for resolving over-subscribed temporal plans. As presented in Chapter 1, over-subscribed situations that involve only temporal constraints have often been modeled by *inconsistent temporal scheduling problems*. A temporal problem is inconsistent if no schedule (Dechter et al., 1991), or execution strategy (Vidal & Fargier, 1999) for problems with uncertain durations, can be found that satisfies all its constraints. For chance-constrained probabilistic temporal problems (Fang et al., 2014), inconsistency means that no strategy for executing its activities exists such that the chance of violating any temporal constraints is lower than the threshold of the chance constraint. To repair an over-constrained temporal problem, one can identify its conflicting constraints, similar to past work on diagnosis, and resolve the conflicts by relaxing one or more of them such that the feasibility of the problem is restored. In addition, since acceptable risk levels may be negotiable in some situations, we can restore the feasibility of chance-constrained problems by identifying constraints that cause the probability of failure to exceed the chance constraint, and increasing the level of accepted risk accordingly.

Several methods have been developed to solve over-constrained temporal problems. In (Beaumont et al., 2001), partial constraint satisfaction techniques were applied to

find a subset of satisfiable constraints. Later, disjunctive constraints and optimality were added in the context of over-constrained Disjunctive Temporal Problems with Preferences (DTPPs) (Peintner et al., 2005). In a DTPP, the disjuncts of every constraint are assigned a preference function that maps the temporal constraint to a cost value. The optimal partial solution is obtained by enumerating consistent subproblems using Branch & Bound, as well as other optimization techniques introduced in (Khatib et al., 2001). Most of the prior work has focused on restoring consistency through complete suspension of constraints, however, in real-world scenarios, the users often want to preserve as much of the schedule as possible to minimize the perturbation.

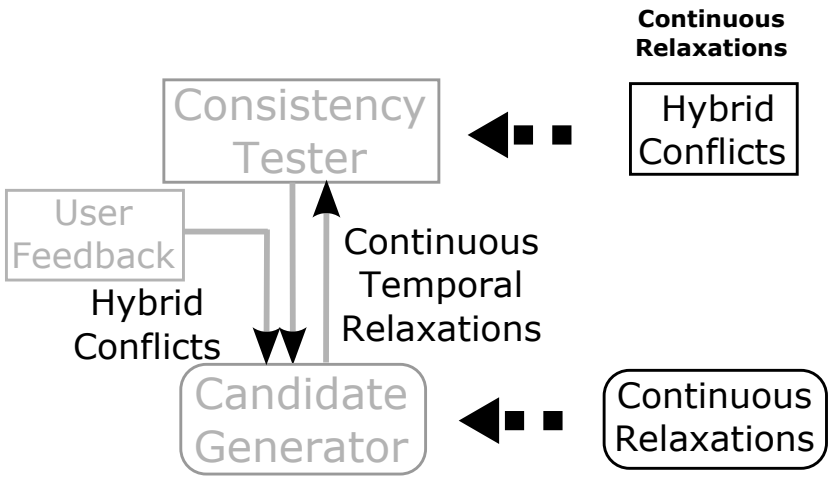


Figure 4-1: Continuous relaxation extensions to BCDR

In Chapter 3, we discussed how BCDR leverages prior work on diagnosis (de Kleer & Williams, 1987; Williams & Ragno, 2002) to compute discrete relaxations. In this chapter, we present the extensions (Figure 4-1) to it for diagnosing conflicts between episodes, and computes continuous temporal relaxations, instead of suspensions, to resolve these conflicts. The key idea behind continuous relaxation is to generalize the discrete conflicts and relaxations, to hybrid conflicts and relaxations, which denote minimal inconsistencies and minimal relaxations to both discrete and continuous relaxable constraints. BCDR is able to generate the most preferred relaxation faster than other state-of-the-art algorithms, and enumerates multiple preferred relaxations

in best-first order, rather than just computing the most preferred relaxation.

The continuous relaxation extension preserves BCDR's iterative approach to discovering conflicting constraints and computing relaxations. We start this chapter with an overview of the continuous relaxation extension, and how it was integrated as part of BCDR's conflict resolution function in Section 4.1. Then in Section 4.2 we discuss the encoding of user feedbacks as hybrid conflicts.

4.1 Computing Continuous Temporal Relaxations

The continuous relaxation extension was first presented in (Yu & Williams, 2013) for enumerating the relaxations to over-constrained temporal problems in best-first order. Once a conflict in a TPN is detected, BCDR uses it to prune the search space by extending each partial candidate with alternative choices, and continuous relaxations for the temporal bounds of episodes in it. An overview of the BCDR algorithm with the continuous relaxation extension is given in Algorithm 5. We will first discuss the changes required in its work flow, then discuss the extensions to the conflict learning and resolution functions in detail.

Similar to the discrete relaxation version presented in Chapter 3, BCDR for continuous relaxation also starts with an empty candidate in the queue (Line 4). A candidate, $Cand$, is now a 6-tuple $\langle A, S, R_e, E', C_r, C_{cont} \rangle$ with assignments A, sequential assignments S, continuous temporal relaxations R_e, additional episodes E', resolved conflicts C_r and continuously resolved conflicts $C_{cont} \subseteq C_r$, all being empty lists in the first candidate. The addition of C_{cont} allows BCDR to track conflicts that are resolved by continuous relaxations in each candidate. BCDR continues looping until the first relaxation is found that makes the input TPN consistent (Line 18). If BCDR does not find a consistent relaxation until the queue is exhausted, it returns null, indicating that no relaxation exists for the input TPN (Line 30).

Within each loop, BCDR first dequeues the best partial candidate, $Cand$ (Line 8). It checks if $Cand$ resolves all known conflicts (Line 9). If not, an unresolved conflict $knownCFT$ will be returned by function ResolveKnownConflicts?, which

Input: A continuously relaxable TPN $Tp = \langle P, Q, V, E, RE, L_e, L_p, f_p, f_e \rangle$.
Output: A solution with continuous relaxation $\langle A, S, R_e, E' \rangle$ that maximizes $\sum_i (f_{pi} - f_{ei})$.

```
1  Cand ← ⟨A, S, R_e, E', C_r, C_cont⟩; //the first candidate;
2  Seq ←GetSeqVariables(P) // the activity sequence variables;
3  Path ← new PATH(P, Seq) // path constraint over all activities;
4  Queue ← {Cand}; //a priority queue that records candidates;
5  C ← {}; //the set of all known conflicts;
6  U ← P; //the list of unassigned controllable variables;
7  while Queue ≠ ∅ do
8      Cand ←Dequeue(Queue);
9      knownCFT ←UnresolvedConflicts(Cand, C);
10     if knownCFT == null then
11         if isComplete?(Cand, U) then
12             newCFT ←PropagatePATH(Cand, Path);
13             if newCFT == null then
14                 AddRoutes(Cand);
15                 newCFT ←TemporallyFeasible?(Cand);
16             endif
17             if newCFT == null then
18                 return Cand;
19             else
20                 C ← C ∪ {newCFT};
21                 Queue ← Queue ∪ {Cand};
22             endif
23         else
24             Queue ← Queue∪ExpandOnVariable(Cand, U);
25         endif
26     else
27         Queue ← Queue∪ExpandOnConflict(Cand, knownCFT);
28     endif
29 end
30 return null;
```

Algorithm 5: The BCDR algorithm

compares the resolved conflicts C_r in $Cand$ with all known conflicts C. $knownCFT$ is then used for expanding $Cand$ by function ExpandOnConflict (Line 27). The child candidates of $Cand$, which includes both discrete and continuous relaxations for the known conflicts, will then be enqueued. For example, assume that we apply BCDR to the example presented in Figure 2-5. When expanding on a partial candidate $\{AM{=}B\}$ with conflict $MS{=}X$, BCDR will create three child candidates that

extends the partial candidate using the two alternative assignments of MS, Y and Z and a temporal relaxation to the upper bound of C_{17} (Figure 4-2). All expanded candidates will then be added back to *Queue*.

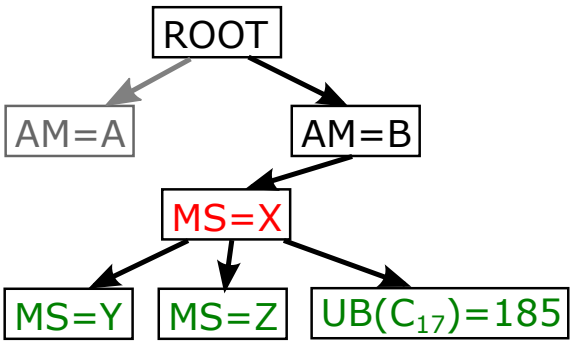


Figure 4-2: Examples of expanding on conflict with continuous relaxation

Finally, BCDR is sound in that every solution returned passes the temporal feasibility check, hence they are guaranteed to be valid. On the other hand, the completeness of BCDR, which indicates if BCDR never misses a valid solution if there exists one, is not as straightforward to prove. It relies on the feasibility checking function to return valid conflicts at all times. We present the detailed proof for BCDR's completeness on computing continuous relaxations in Appendix B.

4.1.1 Conflict Learning From Consistency Checking

Conflict learning is the key for resolving over-subscribed temporal plans. They explain the cause of failure and provide guidance for computing necessary relaxations. Previous approaches (Effinger & Williams, 2005; Li & Williams, 2005), including the discrete relaxation version of BCDR presented in Chapter 3, only extract the set of conflicting episodes and their guard assignments as discrete conflicts.

Definition 21. *A discrete conflict is a pair* $\langle Eps, Guards \rangle$*, where:*

- *Eps is a set of conflicting episodes in the TPN;*
- *Guards is the set of guards assignments for all episodes Eps;*

For example, the discrete conflict from the TPN in Figure 2-5 can be encoded as the following:

Eps: $\{C_{17}$:E-$S \in [0, 180]$; C_7:B_A-$S \in [30, 50]$; C_2:B_L-$B_A \in [45, 60]$;
C_{15}:Y_A-$B_L \in [21, 25]$; C_4:Y_L-$Y_A \geq 65$; C_9:E-$Y_L \in [30, 32]\}$

Guards: $\{AM{=}B;\ MS{=}Y\}$

While computing continuous temporal relaxations, BCDR needs conflicts of higher resolution, since it tries to resolve the conflict by weakening the temporal bounds to the minimal extent. With the discrete relaxation representation, we can only learn about the episodes that are involved in the conflicts, but not the amount of deviation required for their temporal bounds in order to resolve the conflict. Therefore, we define a new representation of conflicts, called **hybrid conflicts**, over the temporal bounds in episodes and their guard assignments.

Definition 22. *A hybrid conflict is a pair* $\langle NCycles, Guards \rangle$*, where:*

- *NCycles is a set of linear expressions defined over temporal bounds of episodes, that forms a negative cycle in the equivalent distance graph of the TPN;*
- *Guards is the set of guards assignments for all episodes in NCycles;*

Each negative cycle in the *NCycles* set is represented by a linear expressions. It represents a necessary constituent of the conflict, and is defined over the lower and upper temporal bounds of episodes, with integer coefficients. BCDR learns new conflicts iteratively from grounded TPNs with different choices made. Given a complete candidate that assigns all active discrete variables, Function TemporallyFeasible? checks the consistency of all activated temporal constraints. The function implements the Bellman-Ford algorithm (Bellman, 1956; Ford, 1956) for checking temporal consistency. If the candidate plan is temporally inconsistent, the algorithm will return a simple negative cycle as the cause of failure. We can extract the minimal inconsistent set of episodes, also called minimal conflict (Liffiton, Moffitt, Pollack, & Sakallah, 2005), using this simple negative cycle. For example, Figure 4-3 shows a simple negative cycle detected in the TPN presented in Figure 2-5: the mission time is too tight for both tasks at site B and Y.

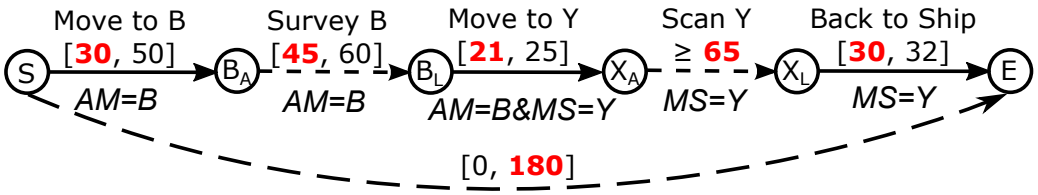


Figure 4-3: A negative cycle in Rich's mission TPN (the guard assignments for each constraint are shown below them)

Given the negative cycle, we can encode a hybrid conflict for it as:

NCycles: {UB(C_{17}) - LB(C_7) - LB(C_2) - LB(C_{15}) - LB(C_4) - LB(C_9);}

Guards: {$AM{=}B$; $MS{=}Y$}

In summary, BCDR learns a **hybrid conflict** from temporally inconsistent candidate plans, which is composed of the temporal bounds of episodes involved in negative cycles and the guard assignments of these episodes. Each negative cycle is encoded by a linear expression over the temporal bounds. Therefore, given the original values of these temporal bounds, the value of this expression must evaluate to negative.

4.1.2 Generalized Conflict Resolutions

Given a hybrid conflict detected in a TPN, we can compute their resolutions and use them to expand our search tree, such that future expansions of the candidates will not enter the infeasible region represented by this hybrid conflict again. This is the core principle behind conflict-directed search. Previous approaches generate discrete relaxations by either flipping the assignments to the discrete variables (Williams & Ragno, 2002; Bailey & Stuckey, 2005) or suspending temporal constraints (Moffitt & Pollack, 2005b). BCDR generalizes the conflict resolution to include both discrete and continuous relaxations: the discrete relaxations deactivate episodes in conflicts by flipping their guard assignments, while the continuous relaxations weakens the temporals bounds of them in order to resolve the conflicts.

Input: A candidate to expand $Cand\ \langle A, S, R_e, E', C_r, C_{cont}\rangle$ and a minimal conflict $knownCFT$.
Output: A set of expanded candidates $newCands$.

```
1  newCands ← {} //collection of newly generated candidates;
2  CFTs ← C_cont ∪ {knownCFT} //conflicts to be resolved continuously;
3  A_alter ← {} //collection of alternative assignments for the ones in knownCFT;
4  for a ∈ knownCFT do
5  |  A_alter = A_alter ∪ GetAlternatives(a);
6  |  for a_l ∈ labels(a) do
7  |  |  A_alter = A_alter ∪ GetAlternatives(a_l);
8  |  end
9  end
10 for a_extend ∈ A_alter do
11 |  if notCompeting(A, a_extend) then
12 |  |  Cand_new ← ⟨A ∪ {a_extend}, S, R_e, E', C_r, C_cont⟩;
13 |  |  newCands ← newCands ∪ {Cand_new};
14 |  end
15 end
16 E_relax ← ExtractConstraints(CFTs);
17 f_obj ← Σ_{e∈E_relax} f_e(Δe);
18 R_new ← Optimize(f_obj, ⟨E_relax ≥ 0⟩);
19 if R_new ≠ null then
20 |  Cand_new ← ⟨A, S, R_new, E', C_r, C_cont⟩;
21 |  newCands ← newCands ∪ {Cand_new};
22 end
23 return newCands;
```

Algorithm 6: Function ExpandOnConflict for hybrid conflicts

Problem 1 (Conflict resolution)**.**

$$\min_{lb'_i, ub'_i} \sum_{i=1}^{|RE|} f_e(lb'_i) + f_e(ub'_i); \tag{4.1}$$

$$s.t.\quad lb'_i - lb_i \leq 0, \qquad ub'_i - ub_i \geq 0; \tag{4.2}$$

$$Conflict_1 \geq 0; \quad Conflict_2 \geq 0; \ldots \quad Conflict_m \geq 0 \tag{4.3}$$

Given a candidate and a hybrid conflict, we extend Function ExpandOnConflict to include two stages in computing resolutions. The first stage is the same as before, which generates resolutions for the conflict by negating variable assignments

(Line 4-15). In the second stage, we compute the optimal continuous relaxation to the relaxable temporal bounds that can resolve the conflict (Line 16-22). We formulate the relaxation as an optimization problem with linear constraints (Line 16) and linear/quadratic objective function (Line 17). The objective function is the minimization over the sum of the relaxation costs of all relaxable temporal bounds, as in (4.1). The variables in this optimization problem are LB'_i s and UB'_i s, which are the relaxable temporal bounds for each episode in the conflicts. The relaxed temporal bounds must be no tighter than the original bounds, as in (4.2). The conflict resolution constraints in (4.3) are added to ensure that all known conflicts are repaired by the resulting continuous relaxations. Given m conflicts, the same number of resolution constraints will be added, each representing the negation of one linear expression in a conflict.

For example, the conflict in (Figure 4-3) involves six temporal bounds. Among them, three are relaxable (C_{17}, C_2, C_4) that can be weakened. We define the following optimization problem for computing the optimal continuous relaxations to resolve this conflict:

$$\begin{gathered} min(f_e(ub'_{C17}) + f_e(lb'_{C2}) + f_e(lb'_{C4})); \\ \text{s.t. } ub'_{C17} - ub_{C17} \geq 0;\ lb'_{C2} - lb_{C2} \leq 0;\ lb'_{C4} - lb_{C4}; \leq 0 \\ ub'_{C17} - lb_{C7} - lb'_{C2} - lb_{C15} - lb'_{C4} - lb_{C9} \geq 0; \end{gathered}$$

The lower bound of C_{17} and the upper bounds of C_2 and C_4 are omitted from the objective function, since they are not part of any constraints in the optimization problem. The solution to the problem is a set of relaxed bounds of C_{17}, C_2, and C_4 that resolves the conflict while minimizing the cost. In this case, the best relaxation is: set lb'_{C4} to 50 and ub'_{C17} to 185. The cost is 27.5. In fact, this problem can also be viewed as a Simple Temporal Problem with Preferences. (Khatib et al., 2001) demonstrates that finding the optimal solution to a STPP with semi-convex preferences is tractable. In real world applications, we may substitute different optimization algorithms, depending on the preference functions, to improve efficiency.

In this example, BCDR will generate three constituent relaxations for the hybrid

conflict: two new assignments derived from flipping guard assignments, and one set of continuous relaxations for the temporal bounds. They are used to extend the partial candidates so that future extensions of it will not run into the same conflict again, as demonstrated in Figure 4-2.

4.2 Incorporating User Inputs as Continuous Conflicts

As demonstrated in Chapter 3, the user can add additional inputs given an unsatisfying solution, or requirements he/she forgot to encode in the original problem. BCDR with the continuous relaxation extension can incorporate them into its search process as a hybrid conflict, such that all future candidate solutions will respect them. In total, given a solution, two types of inputs can be accepted by BCDR:

- Rejection of an assignment, such as "I do not want to visit the methane seeps site X".
- Rejection of a continuous temporal relaxation, such as "The mission duration must be within 4 hours".

BCDR utilizes the conflict-directed approach to efficiently adapt to these inputs. Instead of modifying the input problem and restarting the search process from the beginning, it will record the input as a new conflict and add it to the known conflicts list. The above two types of inputs will be recorded as the following conflicts:

- Rejection of an assignment $X = a$: same as presented in Chapter 3, a new conflict $X = a$ will be created and added to BCDR's conflict collection, such that it will not appear again in any future solutions.
- Rejection of a temporal relaxation $lb'_i = a$ or $ub'_i = b$: a new hybrid conflict with linear expression $lb'_i - a \geq 0$ or $b - ub'_i \geq 0$ will be added to BCDR's conflict collection, such that all future relaxations for lb'_i or ub'_i will be bounded by a and b, respectively.

4.3 Chapter Summary

In this chapter, we presented the second contributions of the thesis: continuous temporal relaxations for over-subscribed temporal plans. The key of our continuous relaxation capability is to generalize the discrete conflicts and relaxations, to hybrid conflicts and relaxations, which denote minimal inconsistencies and minimal relaxations to both discrete and continuous relaxable episodes. Our approach resolves conflicts by weakening the temporal bounds to the minimal extent. With the implementation of an incremental conflict learning and resolution strategy, the algorithm is also able to incorporate user inputs during the process.

Chapter 5

Continuous Temporal Relaxation Under Uncertainty

In this chapter, we present the risk-bounded relaxation extension to BCDR for resolving over-subscribed temporal plans with uncertainty. Instead of weakening the bounds for controllable variables, this extension also resolves conflicts by bounding the outcomes of uncontrollable variables. More specifically, BCDR is able to learn the source of risk through grounding probabilistic temporal durations to set-bounded uncertain duration using risk allocation, and applying controllability checking algorithms to identify conflicting episodes from the grounded plan. Resolutions to these conflicts can then guide us to find feasible relaxations for temporal constraints and/or relaxations for the chance constraint. With the risk-bounded relaxation extension, BCDR is capable of handling temporal plans with the following three types of temporal durations:

- Simple temporal constraints where the duration between lower and upper bounds are controllable. This type is often used for modeling requirements between temporal events. For example, to get home in 40 minutes, given that the subway takes 35 minutes, we know for sure that the requirement can be met.

- Set-bounded uncertain durations, also called contingent constraints (Vidal & Fargier, 1999), involving uncontrolled durations, represented by interval bounds.

Compared to simple temporal constraint, it models the temporal duration as a random variable. The modeler makes a commitment to a degree of robustness by specifying the interval of outcomes to be handled. For example, to get home in 40 minutes, given that the subway takes anytime between 30 and 45 minutes. Unlike the previous example, there is no guarantee that the requirement can be met due to the uncertainty in the subway time: there is a chance that the ride may take more than 40 minutes.

- Probabilistic temporal durations (Tsamardinos, 2002) with information on the likelihood of outcomes. This type is more complex compared to the other two since the uncertainty in duration is accurately modeled using a probability distribution instead of a pair of temporal bounds. In addition, it also allows explicit representation of requirements on risk taken through a chance constraint. For example, we can specify a 95% guarantee that you'll be home in an hour, given that the subway takes a mean time of 30 minutes, with a standard deviation of 5 min.

This chapter is organized as follows. In Section 5.1, we present the extension to the BCDR algorithm that can handle temporal plans with set-bounded uncertain durations (Figure 4-1). We name the extended algorithm BCDR-U, which incorporates a new conflict learning procedure for discovering conflicts with set-bounded uncertain durations. The extension in BCDR-U allows it to restore the controllability, in addition to consistency, of over-subscribed temporal plans. We present two version of the extension, namely BCDR-U(SC) and BCDR-U(DC), for use with strong and dynamic controllability models.

In Section 5.2, we present another extension to BCDR for resolving chance-constrained temporal plans with probabilistic uncertain durations. We name the extension BCDR-C, in which 'C' represents chance constraints. The key idea is to resolve over-subscribed plans by allowing the risk bound, as well as the temporal constraints, to be relaxed continuously. The extension introduced by BCDR-C is a new conflict resolution procedure for handling chance constraints: in addition to relax-

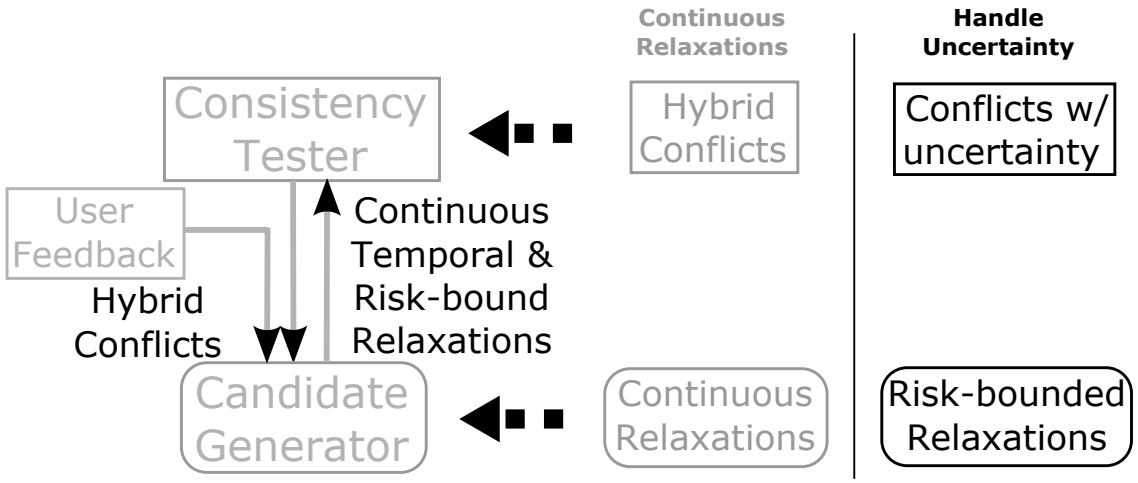


Figure 5-1: Risk-bounded relaxation extensions to BCDR

ing temporal constraints during conflict resolution, it also weakens the risk bounds, which is expressed in the chance constraints. This new conflict resolution step allows BCDR-C to handle overly risky situations by deciding to accept more risk, and achieve a good balance between the risk taken and the temporal requirements for the users. Similar to BCDR-U, we also present two versions of the extension for different controllability models: BCDR-C(SC) for strong controllability and BCDR-C(DC) for dynamic controllability.

Finally in Section 5.3, we discuss a new greedy conflict resolution technique for speeding up BCDR's run-time performance on problems with certain structures. They are essential for the deployment of BCDR in user-facing applications and solving real-world problems of large scale, and are compatible with not only the consistency-based BCDR algorithm, but also its two extensions, BCDR-U and BCDR-C.

5.1 Computing Relaxations for Restoring Controllability

For plans with set-bounded uncertain durations, such as TPNUs, the consistency-based approach cannot find relaxation that is robust to all possible outcomes of the uncertainty. Here, we present the extension for conflict learning and resolution

procedures that accounts for uncontrollable duration, which allows BCDR-U to enumerate strongly and dynamically controllable relaxations for TPNUs. The changes are highlighted in Algorithm 7. First, TemporallyFeasible? checks either strong or dynamic controllability, depending on the type of solution required, and returns a conflict if the candidate fails the test. The key is to learn conflicts from strong and dynamic controllability checking algorithms, and the conflict could be a mixed set of temporal bounds from controllable and uncertain durations. Second, Function ExpandOnConflict is extended to handle conflicts with uncertain durations, whose resolution may involve both relaxing controllable temporal bounds and tightening the outcomes handled over uncertain durations.

Similar to BCDR, every solution returned by BCDR-U passes the strong or dynamic controllability check, hence it is easy to demonstrate its soundness. The proof for the completeness of BCDR-U is not as straightforward, since it also relies on the strong and dynamic controllability checking function to return valid conflicts. The detailed proof for BCDR-U's completeness is presented in Appendix B.

5.1.1 Conflict Learning For Strong Controllability

For TPNs, a conflict is an inconsistent set of temporal bounds from episodes. It can be detected by negative loop detection algorithms: a negative cycle in the equivalent distance graph of a grounded TPN can be mapped to a set of conflicting temporal bounds from episodes. This is because of the one-to-one mapping between the distance edges and the lower/upper temporal bounds of episodes. However, this method does not apply to controllability checking algorithms. Due to the reduction procedures in both strong and dynamic controllability checking, the one-to-one mapping property is not preserved: during reductions, new distance edges are created and added to the graph, and the weights of some edges are modified. We cannot extract the sets of conflicting temporal bounds from the negative loops in reduced graphs directly.

The key to solve this issue is to understand what controllable and uncertain duration contributed to each distance edge in the reduced graph. We name these episodes the *supporting constraints*. The supporting constraints for an edge include the source

Input: **A relaxable TPNU** $Tp = \langle P, Q, V, V_r, E, E_u, RE, RE_u, L_e, L_p, f_p, f_e \rangle$.
Output: Assignments and relaxations $\langle \boldsymbol{A, S, R_e, R_u, E'} \rangle$ that maximizes $\sum_i (f_{pi} - f_{ei})$.

```
1  Cand ← ⟨A, S, R_e, R_u, E', C_r, C_cont⟩; //the first candidate;
2  Seq ←GetSeqVariables(P) // the activity sequence variables;
3  Path ← new PATH(P, Seq) // path constraint over all activities;
4  Queue ← {Cand}; //a priority queue that records candidates;
5  C ← {}; //the set of all known conflicts;
6  U ← P; //the list of unassigned controllable variables;
7  while Queue ≠ ∅ do
8      Cand ←Dequeue(Queue);
9      knownCFT ←UnresolvedConflicts(Cand, C);
10     if knownCFT == null then
11         if isComplete?(Cand, U) then
12             newCFT ←PropagatePATH(Cand, Path);
13             if newCFT == null then
14                 AddRoutes(Cand);
15                 newCFT ←TemporallyFeasible?(Cand);
16             endif
17             if newCFT == null then
18                 return Cand;
19             else
20                 C ← C ∪ {newCFT};
21                 Queue ← Queue ∪ {Cand};
22             endif
23         else
24             Queue ← Queue∪ExpandOnVariable(Cand, U);
25         endif
26     else
27         Queue ← Queue∪ExpandOnConflict(Cand, knownCFT);
28     endif
29 end
30 return null;
```

Algorithm 7: BCDR-U for solving TPNUs

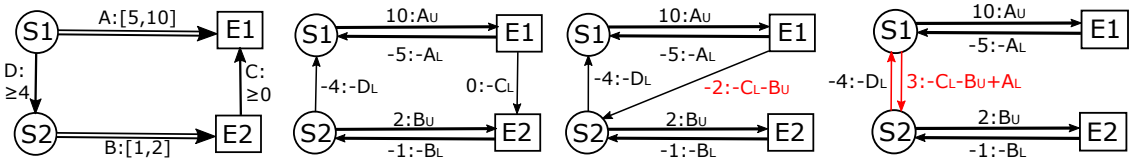
constraint and the constraints that modify the weight of the edge during reduction. We extend the polynomial time algorithm in (Vidal & Fargier, 1999) with additional procedures for recording supporting constraints during reductions (Algorithm 8). This extension enables the algorithm to extract a conflict from a negative loop in the reduced graph. The input to it is a grounded TPNU without any unassigned variables. There are three major steps in this algorithm:

- Map the grounded TPNU to its equivalent distance graph and record the supporting constraints of each distance edge in the graph with its source (Line 1), which is either an upper or lower bound of a temporal constraint.

- Reduce all non-contingent edges that start (Line 14) or end (Line 5) at an uncontrollable node using the triangular reduction rule. If constraint A is reduced to C through B, the supporting constraints of C will be updated to the union of the supporting constraints of A and B (Line 11, 20). A review of the triangular reduction algorithm is presented in Appendix C.

- After the reductions, we run the Bellman-Ford algorithm on the reduced graph (Line 24). If a negative loop is detected, we collect the supporting constraints of all its edges into a set (Line 25) and return it as a conflict that makes the problem uncontrollable. Otherwise, the function returns null to indicate that the input TPNU is strongly controllable.

We demonstrate this process using a temporal network with four constraints (Figure 5-2a): A and B are uncertain durations; C and D are controllable temporal constraints. First, we map the network to its equivalent distance graph (Figure 5-2b). Each distance edge in the graph is labeled with its weight and supporting constraints. The subscript after the constraint name, either U or L, specifies if the distance edge is generated from the upper or lower bound of the constraint.



(a) (b) (c) (d)

Figure 5-2: Supports recording during the triangular reduction for checking strong controllability

There are two non-contingent edges in the graph, S2-S1 and E1-E2, and E1-E2 starts and ends at received nodes (denoted by squares in the graph). We first reduce it using edge S2-E2, a contingent edge that shares the same end node with E1-E2. The

Input: A grounded TPNU $Tp = \langle V, E, E_u, L_e, L_p \rangle$.
Output: A conflict $\langle NCycles, Guards \rangle$ that makes Tp uncontrollable

```
1  DG ←GetDistanceGraph(Tp);
2  ReductionQ ←NonContingentEdges(DG);
3  while ReductionQ ≠ ∅ do
4  |  α ←Dequeue(ReductionQ);
5  |  if End(α) is uncontrollable then
6  |  |  β ← ContingentEdgeEndAt(End(α)) //retrieve the contingent
   |  |  edge that ends at the end node of edge α;
7  |  |  α' ← Reduce(α, β);
8  |  |  γ ←GetEdge(Start(α),Start(β));
9  |  |  if Weight(α') <Weight(γ) then
10 |  |  |  γ ← α';
11 |  |  |  Supports(γ) ← Supports(α, β);
12 |  |  |  ReductionQ ← ReductionQ ∪ γ;
13 |  |  endif
14 |  else if Start(α) is uncontrollable then
15 |  |  β ← ContingentEdgeStartAt(Start(α)) //retrieve the contingent
   |  |  edge that ends at the start node of edge α;
16 |  |  α' ← Reduce(α, β);
17 |  |  γ ←GetEdge(End(β),End(α));
18 |  |  if Weight(α') <Weight(γ) then
19 |  |  |  γ ← α';
20 |  |  |  Supports(γ) ← Supports(α, β);
21 |  |  endif
22 |  endif
23 end
24 NCycle ←Bellman-Ford(DG);
25 return GetSupports(NCycle);
```

Algorithm 8: Strong controllability checking algorithm

result is a new edge E1-S2 with weight -2 and supporting constraints C_L, B_U, which are the union of the supporting constraints of E1-E2 and S2-E2 (Figure 5-2c). Since E1-S2 starts at an received node, we can further reduce it using E1-S1. The result is edge S1-S2 with weight 3 and supporting constraints C_L, B_U, A_L (Figure 5-2d).

It can be seen from the reduced graph that there is a negative cycle of two edges: S1-S2 and S2-S1. The negative cycle indicates that the original STNU is not strongly controllable, and the supporting constraints of these two edges, $\{A_L, B_U, C_L, D_L\}$, are in conflict and cause the failure. The linear expression in the $NCycles$ component of this conflict is $-LB(D) - LB(C) - UB(B) + LB(A)$, whose value is evaluated to -1

without any relaxations to the temporal bounds.

Using this algorithm for checking strong controllability and extracting conflicts does not add much overhead: it takes the same order of magnitude in time compared to consistency checking algorithms. Given a network with V events and E temporal constraints, there will be at most $2E$ reductions and support constraint recordings. The time complexity of strong controllability is thus the same order of magnitude as consistency checking: both are dominated by the $O(VE)$ negative cycle detection.

5.1.2 Conflict Learning For Dynamic Controllability

Our approach for learning conflicts from dynamic controllability checking algorithm is similar to that for strong controllability. We extend the FASTDCCHECK algorithm in (Morris, 2006) with additional steps in its reduction procedures to record the supporting constraints of reduced edges. A review of the original FASTDCCHECK algorithm is presented in Appendix C. Here we present the pseudo code of our revised algorithm that extracts conflicts from uncontrollable problems (Algorithm 9).

As proved in (Morris, 2006), a grounded TPNU, which is equivalent to an STNU if we only consider the temporal constraints from episodes, is dynamically controllable if and only if it does not have a semi-reducible negative cycle. The FASTDCCHECK algorithm is designed based on this theorem. It converts the STNU to an equivalent distance graph of normal form (Line 1) and identifies all negative paths that start with a lower-case edge (Morris & Muscettola, 2005), called *moat paths*, through propagations (Line 6). The input STNU is determined to be dynamically controllable if none of these negative paths leads to a semi-reducible negative cycle (Line 3, 17). The check requires at most K iterations (Line 2), where K is the number of lower case edges in the equivalent distance graph of the STNU.

During the reduction of moat paths, we record the supporting constraints for each reduced edge (Line 9). If the ALLMAXCONSISTENT function, which implements the Bellman-Form algorithm on all non-lower case edges, captures a negative cycle in the reduced graph, it will return a conflict that collects the supporting constraints of all edges in the cycle. There are five types of reductions in this procedure (Morris &

Input: A grounded TPNU $Tp = \langle V, E, E_u, L_e, L_p \rangle$.
Output: A conflict $\langle NCycles, Guards \rangle$ that makes Tp uncontrollable

```
1  DG ←GetNormalDistanceGraph(Tp);
2  for 1 to K do
3  |  NCycle ←AllMaxConsistent(DG);
4  |  if NCycle == null then
5  |  |  for E in LowerCaseEdges(DG) do
6  |  |  |  moatPaths ←Propagate(E);
7  |  |  |  for Path in moatPaths do
8  |  |  |  |  E' ←Reduce(E,Path);
9  |  |  |  |  Supports(E') ← Supports(E, Path);
10 |  |  |  |  AddToGraph(E', DG)
11 |  |  |  end
12 |  |  end
13 |  else
14 |  |  return GetSupports(NCycle);
15 |  endif
16 end
17 NCycle ←AllMaxConsistent(DG);
18 return GetSupports(NCycle);
```

Algorithm 9: Modified FASTDCCHECK algorithm for learning conflicts from uncontrollable networks (changes are highlighted in bold)

Muscettola, 2005; Morris, 2006), and the support recording process is demonstrated for each of them in Figure 5-3.

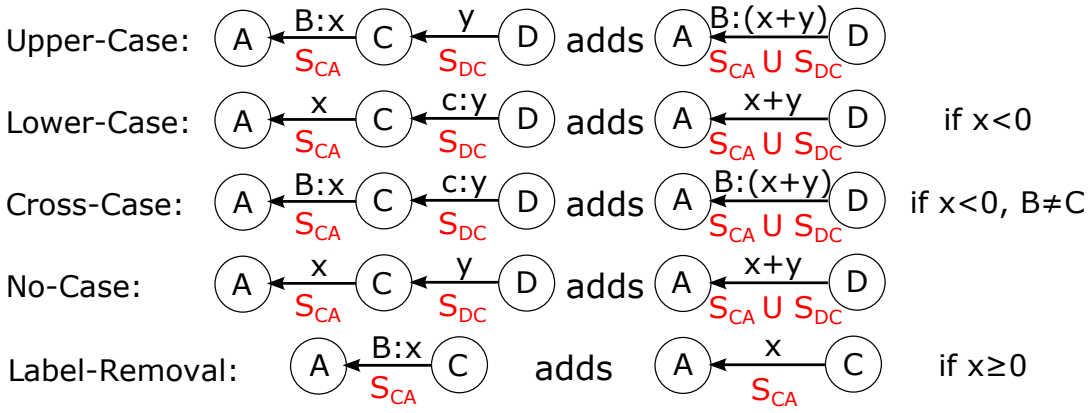


Figure 5-3: Record supporting constraints during the FASTDCCHECK reductions

Next, we demonstrate the conflict learning process using a simple dynamic controllability checking example (Figure 5-4). There are three events, E1, E2 and E3, in this example STNU. These events are connected by two constraints A and B: A

is an uncertain duration with a bound of [10,15], while B is a simple temporal constraint with a bound of [1,1]. The first step of controllability checking is to map the STNU to a normalized form (Morris & Muscettola, 2005), which decouples the lower bounds from each uncertain duration (Figure 5-5). We can then generate the equivalent distance graph using the normalized STNU. Note that each distance edge in the graph, including conditional edges, is labeled with a linear expression over constraint bounds. The expression encodes the source of an distance edge's weight value, such as the example in Figure 5-6.

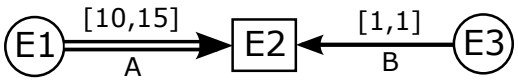


Figure 5-4: The original STNU

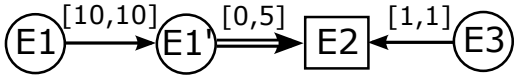


Figure 5-5: The normalized STNU

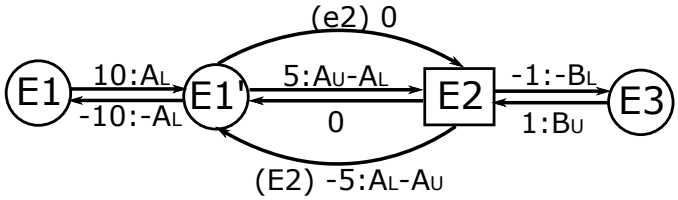


Figure 5-6: The equivalent distance graph of the STNU

The next step is to identify and reduce all moat paths in the distance graph using the iterative method introduced in (Morris, 2006). In this example, there is only one valid moat path: $E1' \rightarrow E2 \rightarrow E3$. This path has a negative weight, starts with a lower-case edge, and can be reduced to a single edge using a lower-case reduction. The reduced edge (represented by a dotted arrow in Figure 5-7) of the moat path has a weight of -1, and is supported by a linear expression, $-B_L$, that combines the expressions of all edges in the moat path.

After all applicable reductions, the final step is to run AllMaxConsistent check on the resulting graph, which checks the consistency of the graph without

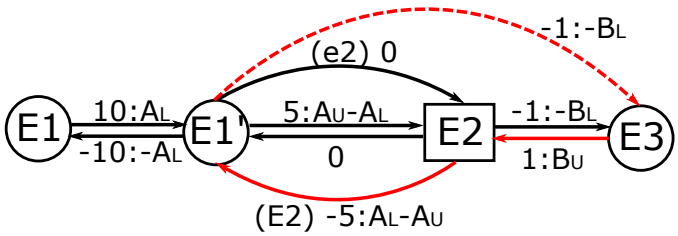


Figure 5-7: The distance graph with a reduced edge

the lower-case edges. It will reveal any negative cycle in the reduced graph, whose existence indicates that the STNU is not dynamically controllable. In this example, one negative cycle can be detected that contains edge $E3 \rightarrow E2$, $E2 \rightarrow E1'$, and the reduced edge $E1' \rightarrow E3$. From this cycle, we can identify the linear expression that caused this conflict from all distance edges in the cycle: $UB(B) + LB(A) - UB(A) - LB(B) \leq 0$. In addition, there is another subtle but necessary element of this conflict: the reduction that adds edge $E1' \rightarrow E3$. The negative cycle would not exist without this reduced edge. Therefore, the expression that supports the reduction, $-LB(B) \leq 0$, which guarantees a negative weight for the moat path, is included in the conflict. The conflict we can extract from the STNU's negative cycle is a conjunction of two linear expressions, $\{B_U + A_L - A_U - B_L; -B_L\}$. Making any one of them non-negative will resolve this conflict.

In summary, learning conflicts from dynamic controllability checking with FASTDCCHECK requires recording the supporting expression for each distance edge and reduction. Once a negative cycle is detected, we can extract a conflict by collecting (1) the expressions for each edge in the cycle; and (2) the expressions required by the reductions that added edges to the cycle. The conflict is a conjunction of these linear expressions, which are all negative and defined over the temporal bounds of constraints.

Currently, extracting conflicts from dynamic controllability checking is significantly harder than that for strong controllability, even though the extra time and space required by recording supports during reductions does not increase the overall complexity of the algorithm. The FASTDCCHECK algorithm is currently the second fastest DC checking algorithm with a complexity of $O(N^4)$, which is an order of mag-

nitude higher than checking strong controllability. To improve run-time efficiency, the algorithm can be terminated and return true after a no-reduction iteration. This is similar to the implementation in (Morris & Muscettola, 2005) and will not affect the correctness of the results. The integration of BCDR with a cubic DC checking algorithm from (Morris, 2014), is expected to further improve performance and is part of our future work to explore.

5.1.3 Resolving Conflicts with Uncontrollable Durations

As described in the beginning of this subsection, both BCDR-U(SC) and BCDR-U(DC) algorithms use the resolutions to unresolved conflicts to expand the search tree. For TPNUs, there are three options for resolving its hybrid conflicts, which may include both controllable constraints and uncertain duration:

- Flipping the guard assignments to deactivate episodes.
- Relax the temporal bounds of episodes with controllable constraints.
- Tighten the temporal bounds of episodes with uncertain durations.

The conflict resolution process for TPNUs, similar to the one for TPNs, is separated into two stages. The first stage is identical to that for TPNs: we look for alternative assignments that can deactivate one or more episodes in the conflict, and use them to generate new candidates.

The second stage implements option 2 and 3. We compute the continuous relaxations to the relaxable temporal bounds in the conflicts. The linear expressions in each conflict's negative cycles provide guidance for ExpandOnConflict to resolve them. A conflict is eliminated if any of its linear expressions is made non-negative. For example, we can resolve the conflict in Figure 5-7 using the following two approaches:

- Set $B_U + A_L - A_U - B_L \geq 0$, e.g. increasing A_L to 15.
- Set $-B_L \geq 0$, e.g. lowering B_L to 0.

Intuitively, to resolve a conflict we can directly require the weight of a previously negative cycle to be non-negative, or we can make sure the reduction which adds an edge never occurs. This choice in conflict resolution is unique to dynamic controllability conflicts: a hybrid conflict from consistency or strong controllability checking only introduces one linear expression. This choice provides more flexibility in conflict resolution, although it also increases the complexity of the problem: to compute the optimal resolutions, BCDR-U(DC) may need to evaluate all possible repairs for all conflicts. The search branches each time BCDR-U(DC) expands on a conflict. If a quick response is desired by the user, BCDR-U(DC) should be implemented with an anytime search strategy.

Once an expression is selected for each conflict, we can again formulate a constraint optimization problem and compute the resolutions using an optimization solver in polynomial time, assuming that the objective function remains semi-convex. There are two categories of variables in the optimization problem: relaxed lower and upper temporal bounds for episodes with controllable constraints (lb'_i and ub'_i) and tightened lower and upper bounds for episodes with uncertain durations (lb'_{uj} and ub'_{uj}). These are given in Problem 2.

Problem 2 (Conflict resolution with set-bounded uncertain durations)**.**

$$\min_{lb'_i, ub'_i, lb'_{uj}, ub'_{uj}} \sum_{i=1}^{|RE \setminus RE_u|} f_e(lb'_i) + f_e(ub'_i) + \sum_{j=1}^{|RE_u|} f_e(lb'_{uj}) + f_e(ub'_{uj}); \tag{5.1}$$

$$s.t. \quad lb'_i - lb_i \leq 0, \qquad ub'_i - ub_i \geq 0; \tag{5.2}$$

$$lb'_{uj} - lb_{uj} \geq 0, \qquad ub'_{uj} - ub_{uj} \leq 0, \qquad ub'_{uj} - lb'_{uj} \geq 0; \tag{5.3}$$

$$Conflict_1 \geq 0; \quad Conflict_2 \geq 0; \ldots \quad Conflict_m \geq 0; \tag{5.4}$$

The constraints in the optimization problem enforce the necessary properties. For TPNUs, we have an additional set of constraints: for lower and upper temporal bound variables of uncertain durations, their value must be within the range defined by the original bounds, and the new lower bound is smaller than the new upper bound,

encoded by (5.3). For example, given the conflict in Figure 5-4, the relaxed bounds for uncertain duration A, lb'_{uA} and ub'_{uA}, must follow $10 \leq lb'_{uA} \leq ub'_{uA} \leq 15$.

5.2 Computing Risk-bounded Relaxations

Finally, we present the third continuous relaxation extension to the BCDR algorithm, called BCDR-C, that allows it to resolve over-subscribed cc-pTPNs. The extension leverages ideas from (Fang et al., 2014) for grounding probabilistic Simple Temporal Problems (pSTPs) into deterministic STNUs, and uses the conflict-directed framework for efficient conflict detection and resolution. Given a cc-pTPN, BCDR-C enumerates feasible solutions in best-first order: a solution is a complete set of assignments, a collection of relaxations for temporal bounds of episodes and chance constraint. Each resolution supports a grounded TPNU whose probability of failure is bounded by the relaxed chance constraint. This requires the conflict-directed approach to support both relaxation and risk allocation: given the grounded TPNU of a cc-pTPN that represents a specific set of choices and risk allocation, BCDR-C will identify the conflicts between episodes and use their resolutions to guide the search towards feasible risk allocation and constraint relaxations. The two key modifications from BCDR-U to BCDR-C are the following:

- First, an additional step of risk-allocation is required for grounding the input cc-pTPN to a TPNU. This allows us to check the feasibility and extract conflicts between episodes using the algorithms developed for TPNUs.
- Second, in addition to flipping assignments and relaxing temporal bounds, the conflict resolution step can also adjust risk allocation over uncertain durations in order to resolve all known conflicts while maintaining the risk taken. Note that this step may require a non-linear optimization solver if the probabilistic distribution of any uncertain duration is non-linear.

We first present an overview of the algorithm that highlights the modifications, then discuss the risk-allocation and chance constraint relaxation procedures in detail.

Input: A cc-pTPN $Tp = \langle P, Q, V, V_r, E, E_d, RE, L_e, L_p, f_p, f_e, \Delta_t, r\Delta_t, f_\Delta \rangle$.
Output: A solution $\langle A, S, R_e, \Delta'_t, N_{alloc}, E' \rangle$ that maximize $\sum_i (f_{pi} - f_{ei} - f_\Delta)$.

```
1  Cand ← ⟨A, S, R_e, Δ_t, N_default, E', C_r, C_cont⟩; //the first candidate with the
   default risk allocation;
2  Seq ←GetSeqVariables(P) // the activity sequence variables;
3  Path ← new PATH(P, Seq) // path constraint over all activities;
4  Queue ← {Cand}; //a priority queue that records candidates;
5  C ← {}; //the set of all known conflicts;
6  U ← P; //the list of unassigned controllable variables;
7  while Queue ≠ ∅ do
8  |  Cand ←Dequeue(Queue);
9  |  knownCFT ←UnresolvedConflict(Cand, C);
10 |  if knownCFT == null then
11 |  |  if isComplete?(Cand, U) then
12 |  |  |  newCFT ←PropagatePATH(Cand, Path);
13 |  |  |  if newCFT == null then
14 |  |  |  |  AddRoutes(Cand);
15 |  |  |  |  newCFT ←TemporallyFeasible?(Cand);
16 |  |  |  endif
17 |  |  |  if newCFT == null then
18 |  |  |  |  return Cand;
19 |  |  |  else
20 |  |  |  |  C ← C ∪ {newCFT};
21 |  |  |  |  Queue ← Queue ∪ {Cand};
22 |  |  |  endif
23 |  |  else
24 |  |  |  Queue ← Queue∪ExpandOnVariable(Cand, U);
25 |  |  endif
26 |  else
27 |  |  Queue ←
   |  |  Queue∪ExpandOnConflictAndAllocateRisk(Cand, knownCFT);
28 |  endif
29 end
30 return null;
```

Algorithm 10: BCDR-C algorithm for solving cc-pTPNs

The pseudo code of BCDR-C is presented in Algorithm 10. Similar to BCDR-U, we implement BCDR-C with a priority queue for enumerating resolutions in best-first order. The algorithm starts with an empty candidate (Line 1) that has no assignments or relaxations, and an empty set of resolved conflicts (C_r and C_{cont}). The candidate is associated with a default risk allocation over all probabilistic uncertain

durations, which is represented by a grounded TPNU ($N_{default}$). The initial allocation is computed from a non-linear solver and is conservative enough to satisfy the chance constraint. The initial candidate is the only element in the queue before search starts (Line 4).

Within the main loop, BCDR-C first dequeues the best candidate (Line 8) and checks if it resolves all known conflicts (Line 9). If not, a conflict $knownCFT$ will be returned by function UnresolvedConflict. The unresolved conflict is then used for expanding $Cand$ (Line 27). All child candidates returned by function ExpandOnConflictAndAllocateRisk resolve $knownCFT$ while satisfying the chance constraints. The function also computes the risk allocation over episodes with probabilistic temporal durations for each candidate, which is added back to the queue for future evaluation and expansion.

If $Cand$ resolves all known conflicts, BCDR-C will proceed to check the controllability of its grounded TPNU (function TemporallyFeasible?, Line 15). BCDR-C(SC) implements this function with a strong controllability checking algorithm, while BCDR-C(DC)'s function implements dynamic controllability checking. If the grounded network passes the check, $Cand$ will be returned as the best resolution to the cc-pTPN (Line 18). Otherwise, a new conflict will be returned by this function and recorded for expanding candidates (Line 20). $Cand$ will also be added back to $Queue$ since it now has an unresolved conflict (Line 21).

Similar to BCDR and BCDR-U, every solution returned by BCDR-C is valid in that they have a feasible risk-allocation and pass the strong or dynamic controllability check, hence it is easy to prove the algorithm's soundness. On the other hand, unlike BCDR and BCDR-U, BCDR-C is not a complete algorithm in that it may fail to return a solution for some cc-pTPNs that do have feasible relaxations. This is a result of its conservative risk allocation procedure. We take a union bound approach when calculating the total risk taken across the temporal bounds allocated for all probabilistic uncertain durations. It guarantees that the solution returned will operate within the specified risk-bound. However, the conservation causes BCDR-C to overestimate the risk taken. As a result, it may not be able to find a feasible risk

allocation for problems with tight risk-bounds, even if one may exist. We will discuss more details on this issue in the following section.

5.2.1 Risk Allocation and Constraint Relaxation

Conflicts provide guidance for BCDR-C to resolve temporally infeasible plans. Given a set of conflicts, we formulate a constrained optimization problem and compute the continuous relaxations using a non-linear optimization solver. To resolve a conflict we can require the weight of any of its linear expressions to be non-negative. There are three categories of variables in the optimization problem: relaxations for temporal bounds of controllable durations (lb'_i and ub'_i), relaxations for chance constraint (Δ'_t), and the allocation of lower and upper bounds for probabilistic uncertain durations (lb'_{pj} and ub'_{pj}). Each category of variables represents a type of conflict resolution: re-allocating risk over probabilistic durations, relaxing the chance constraint, and relaxing controllable temporal constraints. These are given in Problem 3.

Problem 3 (Conflict resolution with chance constraints and probabilistic durations)**.**

$$\min_{\Delta'_t, lb'_i, ub'_i} f_\Delta(\Delta'_t - \Delta_t) + \sum_{i=1}^{|RE|} f_e(lb'_i) + f_e(ub'_i); \tag{5.5}$$

$$s.t. \quad lb'_{pj} - ub'_{pj} < 0 \tag{5.6}$$

$$lb'_i - lb_i \leq 0, \quad ub'_i - ub_i \geq 0; \tag{5.7}$$

$$Conflict_1 \geq 0; \quad Conflict_2 \geq 0; \ldots \quad Conflict_m \geq 0; \tag{5.8}$$

$$\sum_{e_d \in E_d} \textsc{Risk}(lb'_{pj}, ub'_{pj}) \leq \Delta'_t, \quad \Delta'_t \in [\Delta_t, 1) \tag{5.9}$$

The constraints in the optimization problem enforce the necessary properties. For lower and upper bound variables of probabilistic durations, their value can be assigned as long as the lower bound is smaller than the upper bound, encoded by (5.6). For relaxable temporal bounds of controllable durations, their new temporal bounds must

be no tighter than the original bounds, as in (5.7). For controllable durations that are not relaxable, their temporal bounds remain unchanged (omitted from the encoding).

The resolution constraints in (5.8) are added to ensure that all known conflicts are resolved by the relaxations, similar to those in Problems 1 and 2. Given m conflicts, the same number of resolution constraints will be added, each representing one linear expression in each conflict. Finally, we add a risk allocation constraint to ensure that the risk taken meets the chance constraint. This constraint is defined over the lower and upper bound variables of all probabilistic durations. Given distributions of each probabilistic duration and the uncertainty bounds chosen, the RISK function computes the probability mass of the regions outside the uncertainty bounds. BCDR-C uses the union bound to upper-bound the total risk taken across all uncertain durations, as this does not rely on assumptions of independence. If the chance constraint is relaxable, we further require that the relaxed chance constraint is lower bounded by the original chance constraint, and upper bounded by 1. This gives us the flexibility to make trade-offs between risk and performance, if no solution can be found that resolves all conflicts while meeting the current chance constraint. These are described by (5.9).

The objective function, given in (5.5), is defined over f_Δ and f_e for minimizing the cost of temporal and chance constraint relaxations. In the optimization problem, all domain and conflict resolution constraints are linear, while the chance constraint may be non-linear depending on the probabilistic distributions. BCDR-C uses the SNOPT optimization package (Wächter & Biegler, 2006) to solve Problem 3 and compute optimal constraint relaxations and risk allocations. If a solution is returned by SNOPT, function EXPANDONCONFLICT will construct a new candidate with its relaxations for temporal and chance constraints, and the risk allocations. This candidate will then be added as a new branch to BCDR-C's search tree, similar to the process in Algorithms 5 and 7.

Finally, we would like to mention one limitation of BCDR-C on resolving cc-pTPNs. In many real-world scenarios, people may want to impose different chance constraints over different subsets of uncertain durations. The current implementation

of BCDR-C, especially the risk-bounded relaxation procedure, only supports a single chance constraint. It is unable to impose different risk bounds on different sets of episodes while computing new relaxations and risk allocations. Episodes covered by different risk bounds may appear in the same conflict, and it is not clear how to distribute the risk to multiple chance constraints during conflict resolution. The solutions to these issues are part of our future work to explore.

5.3 Implementation Issues and Suggestions

Finally, we discuss two issues revealed during our experiments with BCDR's continuous relaxation extension, and present our solutions to them that will improve the robustness and run-time performance of BCDR for similar types of relaxation problems. The first issue is a numerical instability problem that may cause BCDR to become stuck on a certain conflict: the relaxation generator thinks a conflict has been resolved, while the temporal feasibility checker disagrees and keeps returning the same conflict. Our solution is a parameterized negative cycle detection function whose *sensitivity* can be lowered to match that of the relaxation generator.

The second issue is that the default conflict resolution procedure may be very inefficient for highly over-subscribed temporal plans (plans with a large number of conflicts). As the number of conflicts to resolve increases, the conflict relaxation procedure slows down due to the increasing number of constraints to satisfy. However, much of the computation is not very useful, since the relaxations from a previous iteration become useless when a new conflict is discovered: we have to execute the expensive optimization procedure again with more constraints. Our solution is a mixed greedy-optimal relaxation procedure that uses discrete relaxation when more conflicts are likely to be discovered, and only runs the continuous relaxation procedure if it is likely that no more conflicts may be discovered.

These issues and resolutions may be of particular interest to readers who are applying BCDR to real-world problems with a large set of constraints and highly connected structure. Note that our experimental results of BCDR will be discussed

in the following section: here we focus on the source of these issues and the rationale behind the modification to get BCDR working properly.

5.3.1 Numerical issues in continuous relaxation

The conflict-directed framework used by BCDR and all its extensions follows a generate and test approach, which coordinates the generator (for computing continuous relaxations) and the tester (for checking temporal feasibility) to work together until a relaxation that resolves every conflict is found. When the checker discovers a conflict c, it requires the linear expression c_e of the conflict to be made non-negative. In order to minimize the cost function f_e, the continuous relaxations generated by the optimizer often makes $c_e = 0$. However, in some rare cases, after the arithmetic for the reduction process of checking controllability, the value of c_e becomes $0 - \epsilon$, where ϵ is a very small number. This causes the checker to re-discover the same conflict: the relaxation generator believes that the conflict can be resolved, hence it will not signal failure and tell BCDR to terminate; while its relaxation never satisfies the checker, which causes BCDR to become stuck.

The problem was observed when BCDR-U(DC) ran into an infinite loop. The number of ExpandOnConflict operation counts kept growing as if there is an infinite set of conflicts to resolve. A further investigation into this issue revealed that in these non-terminating scenarios, the conflicts learned by the controllability checking algorithm beyond a certain point are all identical. The continuous relaxation generated by the ExpandOnConflict procedure does not resolve the new conflict, causing the checker to re-discover it again and again. This problem is more often observed on problems with highly connected constraints, which require a large amount of reduction during DC checking and increases the chance of numerical precision issues.

We use the example from 5-7 to demonstrate this problem. Recall that the conflict extracted from dynamic controllability checker for this TPNU has two linear expressions: $\{B_U + A_L - A_U - B_L; -B_L\}$. Assume that the relaxation generator decided to increase A_L to 15.0, which effectively eliminates the uncertainty in it, we will get the

relaxed problem in Figure 5-8. Next, the plan with the relaxation is passed back to the controllability checker for verification. The checker executes the same reduction procedures shown in Figure 5-3, and gets a reduced network (Figure 5-9). However, during the reduction, some of the arithmetic operations may introduce errors and the resulting edge in the network has the incorrect weight of -0.0000000001, instead of 0. As can be seen from the graph, the same negative cycle of Edge $E_3 \rightarrow E_2$,$E_2 \rightarrow E_1'$ and $E_1' \rightarrow E_3$ will be detected by the checker, and hence the same conflict will be returned by it, which puts BCDR-U(DC) into an infinite battle with an already resolved conflict.

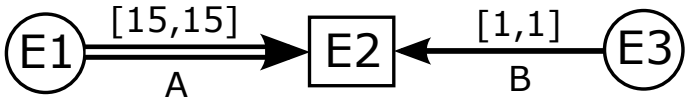


Figure 5-8: The TPNU with a relaxed lower bound for uncertain duration A

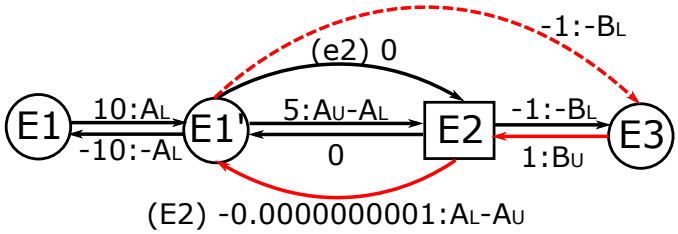


Figure 5-9: The reduced graph with additional edges

There are several options to resolve or reduce the chance of running into this numerical issue: we may slightly relax the *sensitivity* of the controllability checker, set the continuous relaxation generator to *over-relax* a bit, or switch to rational numbers which eliminates the numerical issue completely. We took the first approach since it provides the most intuitive configuration and the flexibility to work in all situations: the over-relaxation approach does not apply to problems whose solution space is very small, such as the RCPSPs in (Cui, Yu, Fang, Haslum, & Williams, 2015); the rational number approach requires an approximation step to scale up the temporal bounds and rounding, whose impact on the correctness of BCDR is difficult to estimate. As mentioned in earlier sections, all temporal feasibility checking functions (consistency, strong controllability and dynamic controllability) depend on a

negative cycle detection function implemented based on the Bellman-Ford algorithm, and the key in our approach is to *loosen* the criteria for a negative loop. Therefore, we modified the condition for distance updates in Bellman-Ford, and the changes are highlighted in Algorithm 11.

Input:
G: $\langle V, E, s\rangle$, a weighted directed graph with vertices V, edges E and source s;
ϵ: the sensitivity settings for negative cycle detection.
Output: *Cycle*: a collection of edges in E that forms a negative cycle.
Initialization:

```
1  Distance ← []; the array for storing minimal distances from source to each
   vertex.;
2  PredecessorEdge ← []; the array for storing predecessor edge for each vertex;
3  // Initialize distance and predecessor array;
   ... ...
4  // Update distances from source to each vertex, only if the distance decreased
   by at least ϵ;
5  for i ∈ [1, LENGTH(V) − 1] do
6  |  for e ∈ E do
7  |  |  if Distance[FROM(e)] + WEIGHT(e) + ϵ < Distance[TO(e)] then
   |  |     ... ...
8  |  |  endif
9  |  end
10 end
11 // Extract negative cycle, if exists;
12 for e ∈ E do
13 |  if Distance[FROM(e)] + WEIGHT(e) + ϵ < Distance[TO(e)] then
   |     ... ...
14 |  |  return Cycle;
15 |  endif
16 end
17 return null;
```

Algorithm 11: Bellman-Ford algorithm with relaxed criteria on distance updates

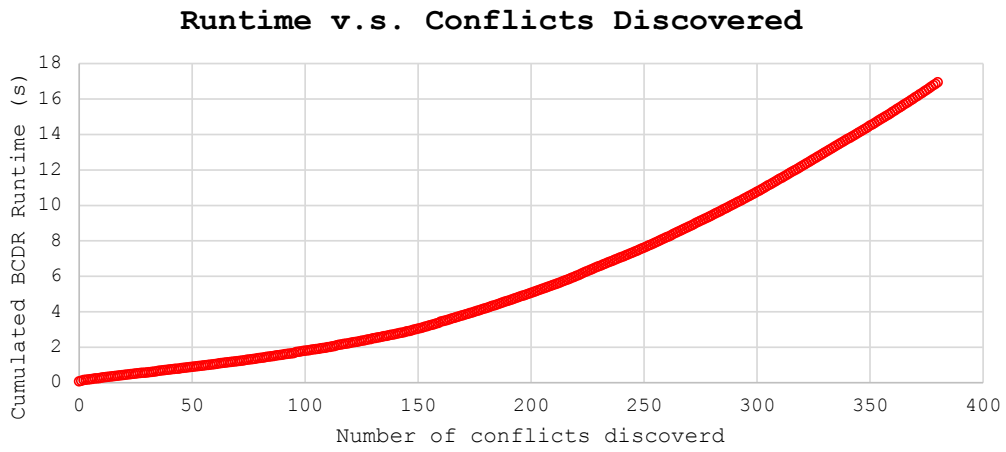
There are three steps in the Bellman-Ford Algorithm: initialization of distance and predecessor for vertices, updating the shortest distances to each vertex from source, and detecting and extracting any negative cycles. The key modification we made is the introduction of a non-negative sensitivity parameter ϵ, which is used during the updates of vertex distances (Line 7) and the extraction of negative cycle (Line 13). It requires the new distance to be ϵ-less than the original distance, instead of just being smaller, effectively making it more difficult for updates to take place.

As a result, no negative cycle with value larger than $-\epsilon$ in the original graph will be detected, since such small differences will not be captured during the distance updates. This parameter can be tuned for different applications to eliminate the possibility of numerical issues while maintaining a good precision and reliability. In our experiments, we set ϵ to 10^{-9} and found it to be sufficient to completely eliminate the numerical issues.

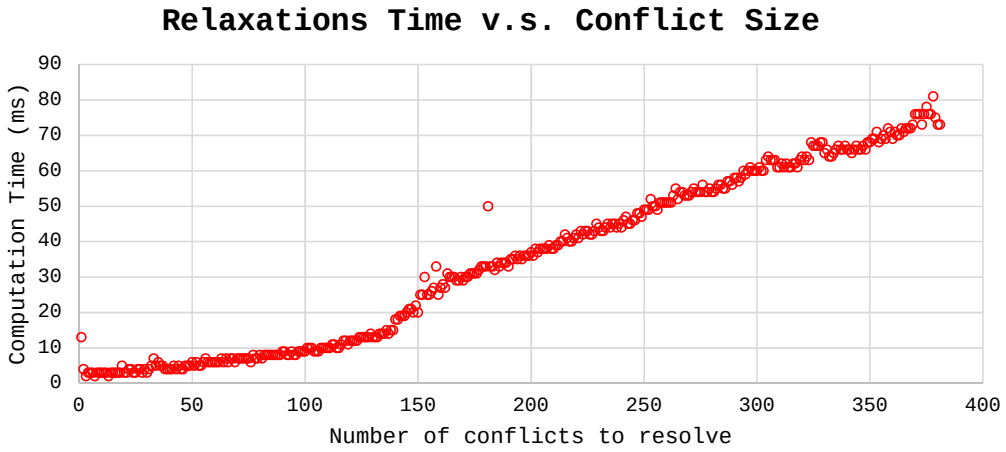
5.3.2 Delayed conflict resolutions

Computing continuous relaxations is the most expensive procedure in BCDR. On the other hand, most of the relaxation computation is not directly contributing to the final solution: we compute relaxations to all known conflicts so that a new candidate can be generated to help find new conflicts. For example, when solving a simple over-subscribed TPN from the train dispatching domain (discussed in the next section) with 417 events and 672 constraints, 381 continuous relaxation operations were executed by BCDR in order to discover the 381 conflicts and find the optimal solution. Out of the 17.01 seconds run-time, 13.14 seconds were consumed by computing continuous relaxations, and the one we are mostly interested in is the last one when we have learned all conflicts. The process is visualized in Figure 5-10, in which we plot the cumulated runtime against the number of conflicts discovered, and the continuous relaxation computation time against the number of conflicts it is resolving. To improve the efficiency of this procedure, the key is to discover new conflicts without incurring so many expensive operations for computing optimal continuous relaxations, such that the run time does not increase as fast when discovering new conflicts.

Therefore, we developed a greedy approach for resolving conflicts during search: picking the relaxable episodes with the lowest relaxation cost in a conflict, and relaxing their lower or upper temporal bounds to the extent that the linear expression of the conflict is non-negative. The exact continuous relaxation procedure only needs to be called to refine the relaxations when a consistent set of greedy relaxations is found: during the search we may use the greedy approach for a new candidate that can push BCDR to discover new conflicts. It takes very little time compared to the



(a)

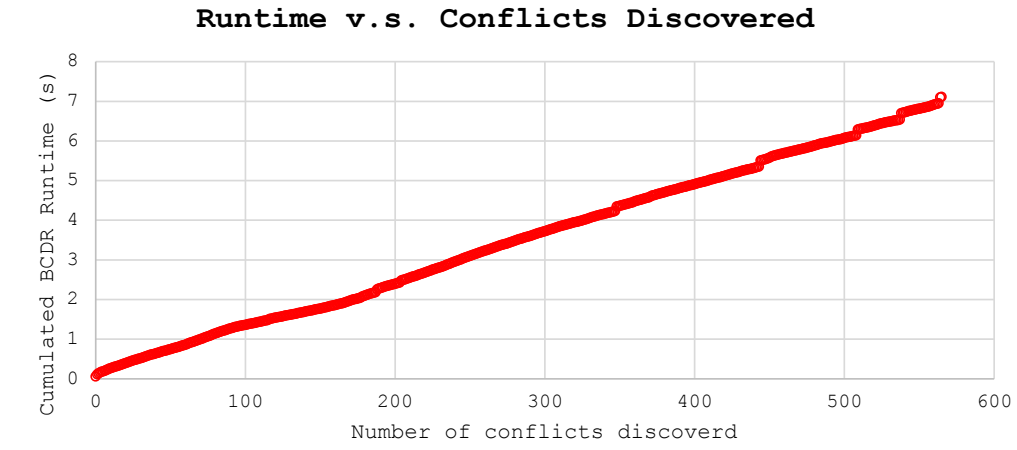


(b)

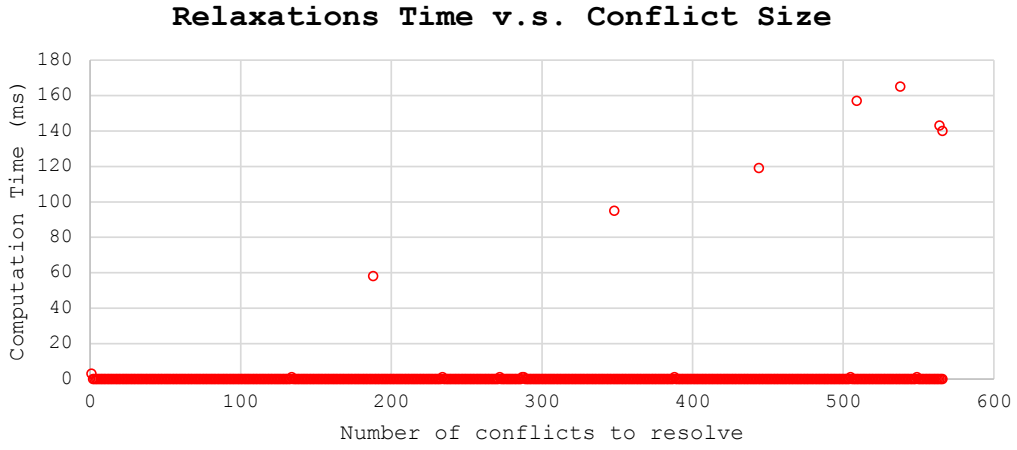
Figure 5-10: Profile of Continuous Relaxations in BCDR Runtime

exact optimal relaxation, since it does not require calling the optimizer. The alternative approach has significantly improved BCDR's runtime performance on large-scale and highly constrained problems. For the same over-subscribed plan, this greedy relaxation approach reduces the runtime to 7.2 seconds, within which only 0.89 second were spent on conflict resolution. The same relaxation time and plot are shown in Figure 5-11. Each point in Figure 5-11b represents the discovery of one conflict. The closer it is to the x-axis, the less time was spent on computing continuous relaxations after discovering the conflict. As can be seen in the graph, less than ten exact relaxations were computed to refine the greedy relaxations on this problem, which greatly reduces BCDR's run time: greedy relaxations were used in most situations to discover

new conflicts. The downside is that we discovered more conflicts than before (566 vs 381), which may not be necessary for generating the optimal relaxation.



(a)



(b)

Figure 5-11: Profile of Combined Greedy/Exact Relaxations in BCDR Runtime

The greedy approach requires very minimum modification to BCDR: instead of formulating and solving the optimization problem in ExpandOnConflict, we compute the greedy relaxations here. Inside ExpandOnConflict, Line 16 to Line 22 (Algorithm 6) are replaced with a new procedure that only looks at the new conflict *knownCFT* (Algorithm 12), instead of all conflicts that were previously resolved continuously. The greedy relaxation procedure iterates through all constraints in the conflict and checks if any constraints involved are relaxable (Line 16-17). If such a constraint is identified, it will relax its bounds, either lower or upper, to the maxi-

mum extent or to the extent that the conflict is eliminated, whichever is smaller (Line 18). The initial value of the conflict's linear expression is captured by the variable $Offset$, and the conflict is resolved once the variable is made zero or negative. If one constraint cannot provide enough deviation, the procedure will move on to the next one, until $Offset$ is made zero or less. If the loop completes but the offset is still positive, it means that no continuous relaxation is available for resolving the new conflict, and the continuous relaxation candidate will not be generated.

```
... ...
Offset ← 0 - EvalExp(knownCFT); amount of relaxation that need to be
applied for resolving the conflict continuously;
... ...
16 for c ∈ knownCFT do
17     if isRelaxable(c) then
18         r = Min(Offset, RelaxationLimit(c));
19         R_new ← R_new ∪ {⟨c, LB(c) − r, 0⟩} or {⟨c, 0, UB(c) + r⟩};
20         Offset = Offset − r;
21     end
22     if Offset ≤ 0 then
23         Cand_new ← ⟨A, S, R_new, E′, C_r, C_cont⟩;
24         newCands ← newCands ∪ Cand_new;
25         break;
26     end
27 end
28 return newCands;
```

Algorithm 12: Modifications to Function ExpandOnConflict (Algorithm 6) for handling candidates with greedy relaxation

The optimal relaxation procedure is moved outside of ExpandOnConflict, and to the main loop of BCDR. Once BCDR has verified the feasibility of a candidate solution, instead of retuning it, an additional procedure will be executed to check if the candidate contains any greedy relaxations (Line 17, Algorithm 13). If true, it will refine the relaxations using the optimization procedure and generate a new candidate. The new candidate will be put back to the queue for further verification, since it may not be the best or consistent candidate after the updates to its continuous relaxation.

This approach works very well on non-conditional plans, whose search tree has one single branch. However, it may not save time on conditional ones. To enumerate

```
... ...
15 newCFT ←TemporallyFeasible?(cand);
16 if newCFT == null then
17 |   if HasGreedyRelaxation(cand) then
18 |   |   cand_optimal ← RefineRelaxation(cand);
19 |   |   Queue ← Queue ∪ {cand_optimal};
20 |   else
21 |   |   return Cand;
22 |   end
23 end
... ...
```

Algorithm 13: Modifications to the BCDR algorithm (Algorithm 5) for greedy continuous relaxation

candidate solutions in best-first order, BCDR needs an accurate estimation for the cost of relaxation in order to prioritize candidates. In order to keep the heuristic function admissible, the cost for all greedy relaxation are set to zero. This is because the greedy relaxations will all have larger than optimal cost, and we have no idea how much the 'minimal cost' will be. Hence, setting the cost to be zero retains the admissible property and still allows BCDR to do a best-first enumeration. However, if we stick to the greedy relaxations, the utility estimation for candidates may be off by a large margin, causing BCDR to waste a lot of time examining candidate solutions that are far from the optimal.

Therefore, we propose a mixed greedy-optimal relaxation approach to achieve a good balance between efficiency and accuracy. The new approach uses greedy relaxation when more conflicts are likely to be discovered, and only runs the continuous relaxation procedure if we are confident that no more conflicts will be discovered, or we have been applying greedy relaxation for more than N times consecutively. If a candidate is found to be consistent, or has more than N greedy relaxations, the exact relaxation procedure will be used. The parameter N is a non-negative integer, and if N is too large then the utility estimation may be off by too much. Therefore, in our experiments we have been using $N = 5$ to achieve a good balance between the time saving and the risk of creating a huge difference in heuristic and actual costs.

5.4 Chapter Summary

In this chapter, we presented the third contributions of the thesis: risk-bounded relaxations under uncertainty. We demonstrated how BCDR's temporal feasibility checker and relaxation generator can be extended to handle plans with temporal uncertainty, either expressed as set-bounded uncertain durations, or probabilistic durations with chance constraints. The two variants, BCDR-U and BCDR-C, are able to diagnose the source of uncertainty in an over-subscribed temporal plan, and enumerate preferred continuous relaxations with bounded risk. The temporal and chance constraint relaxations generated by BCDR resolve conflicts by trading off safety margins with performance, which provides more flexibility for risk management in real world scenarios.

Chapter 6

Domain Relaxation for Parameterized Variables

In this chapter, we present the final contribution of the thesis: computing domain relaxations for over-subscribed temporal plans. The domain relaxation extension (Figure 6-1) enables BCDR to resolve conflicts by weakening the domain constraints on variables, hence allowing additional values to be added to the variable domains. In addition, we introduce a semantic similarity model generated by the word2vec (Mikolov et al., 2013a) and the SemanticMemory (Raiman, 2016) packages to guide the weakening of domain constraints. The similarity model uses high-dimension vector representations of concepts trained on a large corpus of Natural Language data. It has been shown to carry semantic meanings when comparing concepts (Mikolov et al., 2013b), and allows BCDR to prioritize the relaxations for domain constraints.

This chapter is organized as follows. In Section 6.1 we introduce the extension to BCDR for computing domain relaxations for conflicts in over-subscribed temporal plans. In section 6.2 we describe how BCDR interacts with the relaxation generator and semantic similarity model to enumerate domain relaxations in best-first order.

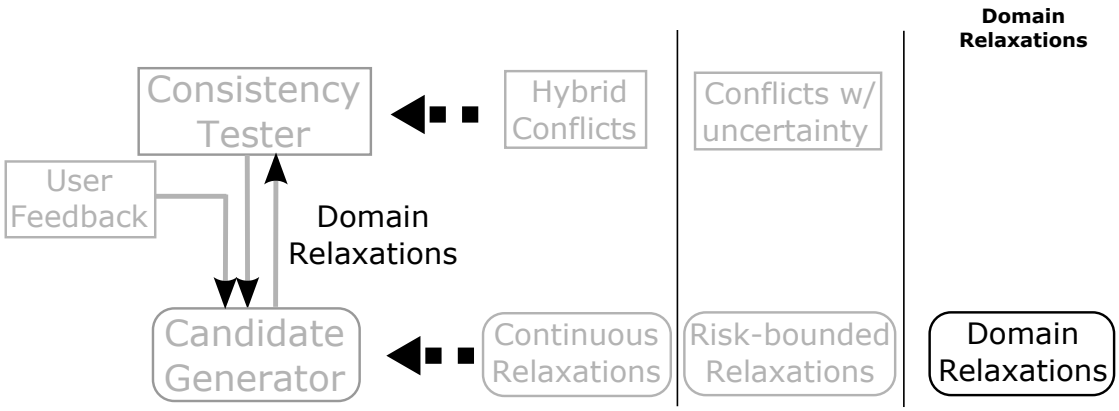


Figure 6-1: Domain relaxation extensions to BCDR

6.1 Computing Domain Relaxations

Given a TPN that encodes all activities and constraints from the users, BCDR fills in the details by computing a set of temporally feasible choices of activities, adding contingencies for likely delays during transit and compute relaxations for the temporal and domain constraints, if necessary. As presented in Chapter 3, BCDR leverages ideas from conflict-directed diagnosis and relaxation algorithms in the literature: it uses the conflict-directed framework from (Williams & Ragno, 2002) for efficient conflict detection and resolution, and generalizes methods from the continuous temporal relaxation techniques in Chapter 4, for enumerating both discrete and continuous relaxations simultaneously. We first present an overview of BCDR with the domain relaxation extension, then discuss in details the two new features: computing and enumerating domain relaxations.

Algorithm 14 presents the pseudo code of BCDR's main function, which introduces the following modifications from BCDR in Chapter 4 to support domain relaxation (highlighted in bold):

- First, in addition to temporal relaxations, the conflict resolution step (Line 30, ExpandOnConflit) was extended to also compute possible domain relaxations to variables in the conflict.

- Second, an additional step of relaxation expansion is added for candidates with partially initiated domain relaxations (Line 9, ExpandDomainRelaxation). Dur-

Input: A domain relaxable TPN $Tp = \langle P, Q, V, E, RE, L_e, L_p, f_p, f_e, P_s, L_s \rangle$.
Output: A solution, $\langle A, S, R_e, R_d, E' \rangle$ that maximizes $f_p - f_e$.

```
1  Cand ← ⟨A, S, R_e, R_d, E', C_r, C_cont⟩; //the first candidate;
2  Seq ←GetSeqVariables(P) // the activity sequence variables;
3  Path ← new PATH(P, Seq) // path constraint over all activities;
4  Queue ← {Cand}; //a priority queue that records candidates;
5  C ← {}; //the set of all known conflicts;
6  U ← P; //the list of unassigned controllable variables;
7  while Queue ≠ ∅ do
8      Cand ←Dequeue(Queue);
9      if ExpandDomainRelaxation(Cand, Queue) then
10         Continue;
11     endif
12     knownCFT ←UnresolvedConflicts(Cand, C);
13     if knownCFT == null then
14         if isComplete?(Cand, U) then
15             newCFT ←PropagatePATH(Cand, Path);
16             if newCFT == null then
17                 AddRoutes(Cand);
18                 newCFT ←TemporallyFeasible?(Cand);
19             endif
20             if newCFT == null then
21                 return Cand;
22             else
23                 C ← C ∪ {newCFT};
24                 Queue ← Queue ∪ {Cand};
25             endif
26         else
27             Queue ← Queue∪ExpandOnVariable(Cand, U);
28         endif
29     else
30         Queue ← Queue∪ExpandOnConflict(Cand, knownCFT);
31     endif
32 end
33 return null;
```

Algorithm 14: An overview of BCDR with domain relaxation extension

ing conflict resolution, BCDR only computes partial domain relaxation candidates, which are not initiated until being dequeued. It allows BCDR to delay the knowledge base queries and semantic similarity comparison, which are computationally expensive operations.

BCDR is also capable of handling uncertain temporal durations in the problem, using an alternative temporal conflict learning and resolution functions presented in Chapter 5. When configured with the corresponding controllability checking and conflict extraction functions for TemporallyFeasible?, such as strong and dynamic controllability, BCDR is able to check controllability and compute risk-bounded relaxations for plans with uncertain durations.

6.1.1 Resolving Conflicts using Domain Relaxation

ExpandOnConflict (Algorithm 15) expands the search tree using new candidates computed from the resolutions to known conflicts. Two options have been used by prior approaches in this procedure: (1) flipping the assignments to deactivate episodes (Chapter 3), and (2) relaxing the temporal bounds of episodes (Chapter 4). For (1), GetAlternatives collects all alternative domain values for assignments in the conflict, and uses the ones that are not competing with any existing assignments (NotCompeting) to generate new candidates. For (2), ContinuouslyRelax evaluates the temporal bounds in the conflict, and weakens some relaxable ones to resolve it. The domain relaxation extension introduces a third option: enlarge the variable domain using similar values. This requires an additional step in the conflict resolution function (Line 5-7): if the discrete variable involved in the conflict has a relaxable domain, a special assignment for the variable, called *SthElse* (something else), will be added in addition to alternatives already encoded in its domain. This special assignment serves as a placeholder that resolves the conflict, and will be expanded with grounded values later to create new candidates.

When BCDR dequeues a candidate, Function ExpandDomainRelaxation (Algorithm 16) will first check if it has any such special assignment (Line 2). If so, it will iterate through all relaxable triples in the variable's domain constraint (Line 3), and extract alternative objects from the knowledge base for them (Line 4). Next, given all alternative options (Q_{alt}) for a triple, Function GetSimilar selects the one that has the highest similarity score to the original one, but have not been used before in R_d. BCDR then queries the knowledge base with the new triple q_{sim}, and generates new

domain assignment a_r and constraints E_r for each of the additional values. Finally, a new candidate is created from each new assignment and then added to the queue (Line 8).

Input: A candidate *cand*: $\langle A, S, R_e, R_d, E', C_r, C_{cont}\rangle$ and an unresolved conflict $currCFT$.
Output: A set of candidates $Cands$ that resolves $currCFT$.

```
1  Cands ← {};
2  for a ∈ A do
3  |  A_alter = A_alter ∪ GetAlternatives(a);
4  |  A_alter = A_alter ∪ GetAlternatives(guard(a));
5  |  if IsRelaxable(variable(a)) then
6  |  |  A_alter = A_alter ∪ {variable(a) = SthElse};
7  |  end
8  end
9  for a_alt ∈ A_alter do
10 |  if NotCompeting(A, a_alt) then
11 |  |  cand_new ← ⟨A ∪ {a_alt}, R_e, R_d, E', C_r, C_cont⟩;
12 |  |  Cands ← Cands ∪ {cand_new};
13 |  end
14 end
15 Cands ← Cands ∪ {ContinuouslyRelax(cand, C_cont ∪ {currCFT})};
16 return Cands;
```

Algorithm 15: Function ExpandOnConflict

We demonstrate this procedure using the travel example from Section 2.5. One conflict in Simon's plan is that he cannot go to restaurant Magic Wok and watch Joy at AMC 16 while arriving home before 9:30pm (Figure 6-2). ExpandOnConflict generates four new candidates from resolutions to this conflict: two from alternative domain assignments in variable *Dinner* and *Movie*, one from continuous relaxation for the trip duration, and one from domain relaxation for variable *Dinner*. After BCDR has evaluated all other candidates and found them to be infeasible or rejected by the users, it dequeues the domain relaxation candidate $Dinner \leftarrow SthElse$ and computes grounded options for it. ExpandDomainRelaxation takes in this special assignment and extracts the relaxable triple of cuisine in the variable's domain constraint. It then identifies the most similar alternative object, Korean, for the original triple with Chinese. Finally, using the new cuisine triple, three new options for

Input: A candidate $cand$: $\langle A, S, R_e, R_d, E', C_r, C_{cont}\rangle$, and the search queue $Queue$.
Output: A boolean value indicating if special assignment $SthElse$ was detected.
Algorithm:

```
for a ∈ A do
    if a == SthElse then
        for q ∈RelaxableTriples(var(a)) do
            Q_alt ← GetOptions(q, var(a));
            q_sim ← GetSimilar(Q_alt, var(a), R_d);
            for ⟨a_r, E_r⟩ ∈QueryKB(var(a), q_sim) do
                cand_new ← ⟨A \ {a} ∪ {a_r}, R_e, R_d ∪ {q_sim}, E' ∪ E_r, C_r, C_cont⟩;
                Queue ← Queue∪{cand_new};
            end
        end
        return True;
    end
end
return False;
```

Algorithm 16: Function ExpandDomainRelaxation

Dinner, Sunny Bowl (SB), Bimbibowl (Bb) and Jang Su Jang (JSJ) were found by querying the knowledge base, and used to resolve the conflict and further expand the search tree.

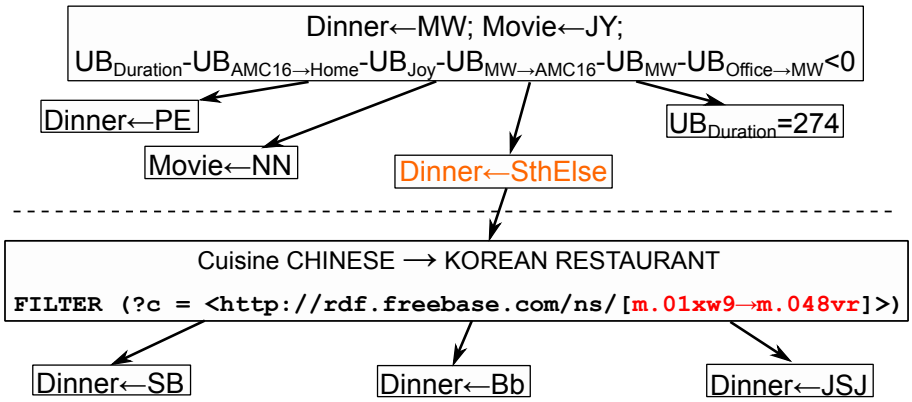


Figure 6-2: Expansion with temporal and domain relaxations

One domain constraint relaxation may be used multiple times for resolving conflicts detected on different branches of the search tree. Instead of querying the knowledge base each time we need to relax the variable's domain, we record the computed relaxations, organize them in an array, and reuse them instead of generating new ones.

The pseudo code of this procedure, which is implemented as part of the relaxation expansion function, is presented in Algorithm 16.

6.1.2 Integration with Knowledge Base

BCDR integrates with a large-scale knowledge base to access the world knowledge, such as restaurants, movies and showtimes, for computing domain relaxations. This knowledge base is constructed from a combination of data sources of content using an ingestion pipeline (Noessner, Martin, Yeh, & Patel-Schneider, 2015). It transforms the raw content into RDF triples and performs entity resolution to merge duplicate entities across different content sources. The resulting knowledge base can be viewed as a very large knowledge graph where the nodes represent entities and the edges represent semantic relations between these entities. The entities are typed, and a proprietary subsumption hierarchy is used to organize these types. The semantic relations have domain and range constraints, and also capture inverse relationships. This knowledge graph can be efficiently accessed and queried via SparQL.

Due to the size of the knowledge base, it is deployed on a dedicated server and BCDR communicates with it through a web API. On average, one query takes 500 ms to complete, which is very significant compared to computing temporal relaxations. While implementing BCDR with domain relaxation extension, it is important to delay the expansion of domain relaxation candidate as much as possible. Therefore, we only do the expansion (ExpandDomainRelaxation) after a candidate is dequeued.

6.2 Prioritizing Domain Relaxations

Given a set of alternative constraints, GetSimilar in Algorithm 16 calculates the similarity scores between them and the object in the original constraint, and then returns the most similar one. For example, given that no Chinese restaurant fulfills both Simon and Christian's requirement, we would like to try Korean cuisine first instead of Mexican or American, since Korean and Chinese are both Asian cuisines and more likely to be preferred. BCDR measures the similarity using a vector repre-

sentation of words, first proposed in (Mikolov et al., 2013a): each object (represented by its Freebase MID) is associated with a 1000-dimension vector of numbers. The vector representations are learned by neural network model based on a large corpus of natural language data, which has been shown to capture semantic properties well (Mikolov et al., 2013b). The similarity score between two objects is computed using the cosine similarity between their vectors: higher scores mean they are more similar. The vector model of BCDR is trained by the continuous skip-gram algorithm in the *word2vec* package with a Google News dataset (Word2Vec, 2013). For example, Figure 6-3 presents the semantic similarity scores between a few alternative cuisines and movie genres in Simon's problem. Out of four alternative cuisines to Chinese, Korean and Thai are closer in distance, with similarity scores of 0.7140 and 0.6945, while Mexican and American cuisine are further away with scores of 0.5169 and 0.3183, respectively. With the semantic similarity measurement, BCDR is able to explore the domain relaxations in best-first order, like prior approaches did for temporal relaxations.

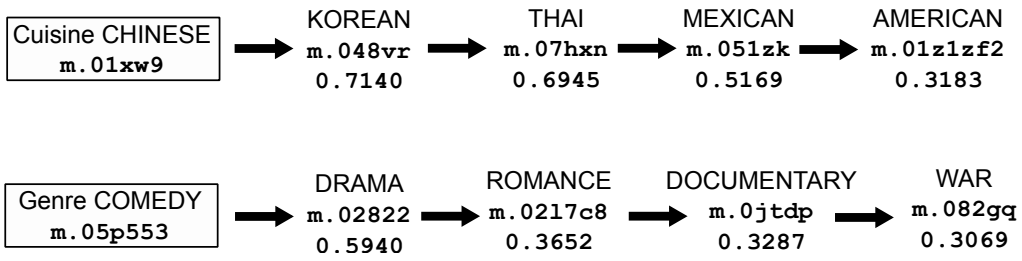


Figure 6-3: Semantic distances between cuisines and genres

The accuracy of word2vec's distance measurement is dependent on the amount of data used in training: it is often more reliable for domains that are commonly encountered in web contents and publications. In addition, the distance measurement is an indication of commonsense, and does not reflect the difference in individuals' perceptions. For example, some people may find Vietnamese food are very different from French food due to the geographic separation of the two countries, while some may find them to be quite similar due to the French influence on Vietnamese cuisine. The word vector model currently used by BCDR only covers about 1.4 million out

of 50 million topics in Freebase due to the corpus used in training. When asked to compare with an object with an undefined vector, the similarity measurement will return zero, which greatly reduces the chance that this object will be considered in relaxation. This limitation was observed during our experiments, and we are working on a backup measure with better coverage to address this issue.

Finally, as the cost functions defined over temporal relaxations and domain relaxations are not compatible (weighted linear cost vs. cosine distance and conditional probability), we normalized the cosine distance by computing its inverse (1/distance) and used it as the cost for domain relaxations.

6.2.1 Alternative Preference Model for Single-Valued Domains

In some scenarios, the activities specified by the users may not be a collection of candidates. Instead, they want to visit a specific restaurant or grocery store. For example, in Simon's evening trip example, he may only prefer to visit the restaurant Magic Wok, instead of any Chinese restaurant. In this case, the domain constraints for the variable *Restaurant* will simply be one equality relation, such as the following SparQL query:

- **Chinese Restaurant**: SELECT ?r WHERE{
  ?r ns:type.object.type ns:dining.restaurant.
  FILTER (?r = <Magic Wok>).}

In such situations, the word2vec model is not able to help prioritize alternative relaxations, since it was trained only on general concepts representing the classes of candidates, not on individual instances. Therefore, to support the relaxation for such domains, we integrated an alternative preference model, called *Semantic Memory*, which operates on individual instances and supports the comparison of them. First presented in (Raiman, 2016), Semantic Memory uses an ontology to keep track of historical user behavior and rank candidates using the discovered preference. Semantic Memory works directly with the description of individual instances, whether it is a restaurant, grocery store or a gas station. It builds an ontology, called a *User*

Profile, over the candidate options from categorized (e.g. rating, cuisine, genre) and uncategorized language data (e.g. reviews). It also supports model updates through interactions with the users, reasoning about preferences hierarchically, and dynamic acquisition of new choices.

The implementation of Semantic Memory includes two components:

- Reader Module: the Reader Module uses a novel topic model, also presented in (Raiman, 2016), to obtain vector representations of choices from their associated textual descriptions and categorized metadata. The resulting vectors are used to build and update the User Profile.

- Memory Module: the Memory Module builds the User Profile, an ontology for each user the system keeps track of. The ontology is constructed as a weighted tree with all possible options available for a variable domain as leaves. The relationship between child and parent nodes in the ontology follow an *is-a* relationship: neighbors and parents are thereby semantically similar. At each leaf node a label holding the user's preference can be in three states: likes, dislikes, unknown. The Semantic Memory uses these labels to rank the options for BCDR during relaxation.

To support the update of User Profile, BCDR will also forward the rejection from the users on a candidate solution to the Semantic Memory package. The update allows the User Profile for the user to be updated for future use, which will order the candidates differently the next time BCDR gets into the same situation (Figure 6-4).

We use the same example from Section 4.2 to demonstrate the behavior of BCDR working with Semantic Memory (Figure 6-5). Assume that there are four additional restaurants available for selection, in addition to *Magic Wok*, which does not meet Simon and Christian's requirements. Semantic Memory organizes them into a tree using their vector representations. It then computes the commute distance from the other four restaurants to Magic Wok using the inference procedures described in (Kemp & Tenenbaum, 2008). First presented to model humans' structural learning of concepts, the commute distance is defined on graphs for making predictions about the

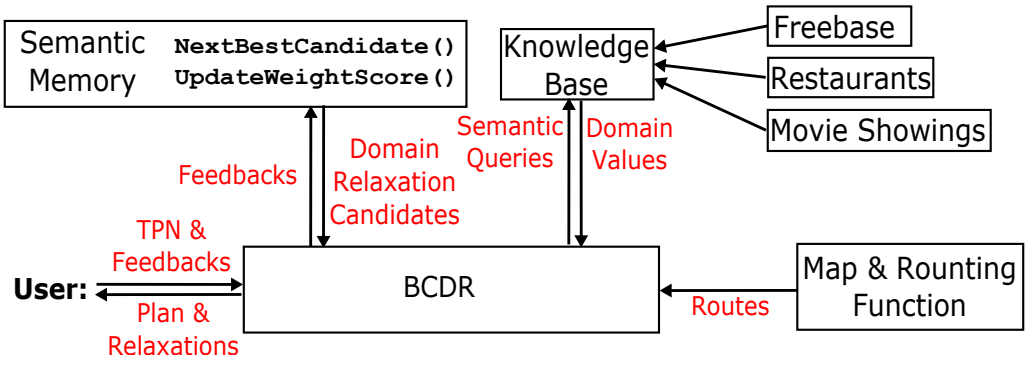


Figure 6-4: The interactions between BCDR and Semantic Memory

similarity between concepts: given that leaf node X is in category A, it indicates how likely is leaf node Y also in category A. It takes into account of not only the distance between graph nodes, but also the number of different paths connecting them. Here, the concept of being in a *category* represents being preferred by Simon. We set the node representing Magic Wok to have value 1 (given that he asked for it from the beginning). Using the tree+diffusion model from (Kemp & Tenenbaum, 2008), we can compute the covariance matrix corresponding to the tree, which can then help us estimate the likelihood that the states of the other four leaf nodes being 1. As shown in the figure, having both A and B to be preferred by Simon is more likely than the other three restaurants.

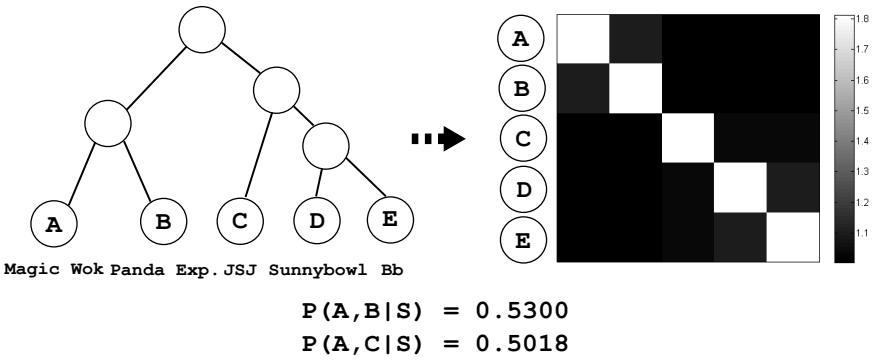


Figure 6-5: Domain Relaxation with Semantic Memory (S represents the matrix that encodes the tree structure)

Compared to word2vec, which has a very wide coverage with a model trained on a large corpus, Semantic Memory requires a much smaller but detailed training

set. It has a limited coverage, while in return, provides much higher resolution of the recommendations, and is designed to be personalized for different users. It is a good complement for the word2vec model in uncommon domains or domains with too many candidates. On the other hand, the distance measure returned from Semantic Memory is a conditional probability, which is not the same as the cosine distance from word2vec, even though they are in the same range. The integration of the two different types of distance measure is part of our future work to explore.

6.3 Chapter Summary

In this chapter, we presented the fourth and final contribution of this thesis: computing domain relaxations for over-subscribed temporal plans. In addition to continuously relaxing temporal bounds, we extend the BCDR algorithm to also compute relaxations for variable domains. Domain relaxation allow more options to be added to the plan for resolving conflicts. BCDR is able to simultaneously enumerate both temporal and domain relaxations in best-first order, and has been integrated with a knowledge base and a semantic similarity calculator for finding good relaxation candidates.

Chapter 7

Uhura: An Advisory System for Resolving Over-Subscribed Travel Plans

Finally, we discuss the design and implementation of Uhura, a dialog-based travel plan assistant that builds upon BCDR and its extensions. Uhura is able to handle over-subscribed plans involving multiple agents, interrelated goals, and efficiently compute continuous temporal, risk-bounded and domain relaxations. Uhura is akin to Siri and Google Now, two of the popular mobile personal assistants that have been used by many people for their day-to-day activities. For example, we use them to send messages, check weather, and find restaurants. Many of these assistants support both verbal and visual communications to make the interaction simpler. However, their functions are limited to simple commands and information retrieval tasks. None of them can understand user requests involving multiple goals and activities, which require planning and scheduling capabilities to properly fulfill. In addition, they lack the capability of identifying and managing uncertainty in planning the activities, or find alternative plans if the current one does not work any more.

There has been much research on building advanced end-to-end personal assistants, but most of these assistants also do not support planning. For example, (Yorke-Smith, Saadati, Myers, & Morley, 2012) reports an end-to-end personal assistant

framework, with a focus on proactivity and task management. (Yeh, Ramachandran, Douglas, Ratnaparkhi, Jarrold, Provine, Patel-Schneider, Laverty, Tikku, Brown, Mendel, & Emfield, 2015) and (Liu, Cyphers, Pasupat, McGraw, & Glass, 2012) report end-to-end personal assistants for TV program discovery, which is able to process user requests involving single goal and activity. (Freed, Carbonell, Gordon, Hayes, Myers, Siewiorek, Smith, Steinfeld, & Tomasic, 2008) describe a personal assistant that helps reduce email overload, and the human-machine collaborative planning prototype system developed by (Allen & Ferguson, 2002) supports planning for a small set of temporal and spatial constraints.

Uhura is able to work with temporal plans of much larger scale and compute relaxations if the plan is over-subscribed. As demonstrated in previous chapters, Uhura provides the following features to make the plan resolution process much easier for the users: 1) natural language communication; 2) mixed initiative goal-directed interaction; 3) support for multiple activities and constraints; and 4) being robust to temporal uncertainty. These capabilities are supported by a coordinated system of three major components, and its architecture is shown in Figure 7-1. The three core components of Uhura are Dialog Manager, BCDR, and Knowledge Base. The dialog manager handles the interactions with users through natural language understanding and generator components, and elicits their goals and requirements as a TPN. It also takes the domain constraints expressed by the user as well as the temporal and spatial constraints, and formulates queries for the Knowledge Base. The results of these queries ground the activities in the TPN with additional episodes and constraints. Finally, the expanded TPN is sent to BCDR, which produces temporally feasible plans and relaxations that best meet the users' requirements.

All reasoning components of Uhura, including BCDR, the knowledge base and the semantic similarity calculator, are implemented as web services. Each time a user provides a new trip related activity or constraint, the dialog manager will decode it, query the knowledge base's API for grounded options, and incorporates them into a TPN for the user's trip. Once the user finishes providing trip information, the dialog manager send a plan query containing the TPN to BCDR, which will then search

for alternative plans and relaxations that best meet the requirements in the TPN. Once a feasible plan and set of relaxation is found, it will be sent back to the dialog manager and presented to the user, both using verbal description of the choices and relaxations made.

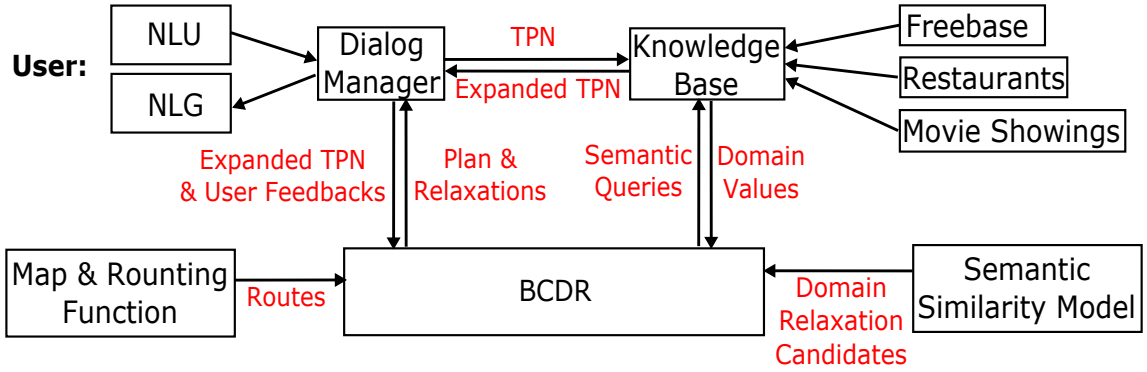


Figure 7-1: The architecture graph of Uhura

In the rest of this chapter, we will focus on the explanation of how we integrate the dialog manager with the BCDR and the knowledge base, to support Uhura's capabilities. The dialog manager and knowledge base presented in our discussions are implemented based on (Ortiz & Shen, 2014) and (Noessner et al., 2015), respectively. But with minimal changes to the input and output languages, the interfaces we introduced can be used for integrating BCDR with other dialog management and point-of-interest database systems.

7.1 Architecture for Integration

As can be seen from the architecture graph (Figure 7-1), the three major components of Uhura have different responsibilities and applications. The key to an effective integration is to **decompose** the overall problem properly, assign the subproblem to the component that has the right reasoning capability, and supply them with the right set of data. Inside Uhura, the dialog manager is responsible for interacting with the users and capturing the planning problems from them. It creates and assigns subproblems that require temporal and spatial reasoning to BCDR, and subproblems

that require semantic reasoning to the knowledge base. In this section, we present the interfaces between three components for creating the temporal plans, evaluating their temporal feasibility and generating relaxations.

7.1.1 BCDR/Dialog Manager – Knowledge Base Interface

Both dialog manager and BCDR need access to background knowledge in order to ground the various activities that meet user's descriptions so that BCDR can generate a satisfactory activity plan for the user. BCDR accepts TPNs with domain constraints specified in arbitrary semantic queries, the dialog manager encodes all the event constraints in first-order logic represented by a semantic graph, while the knowledge base used by us accepts only SparQL as its query language. As presented in Chapter 6, BCDR is able to work with domains specified SparQL while computing relaxations. However, the semantic graph generated by the dialog manager cannot be mapped to SparQL queries directly. Additionally, the semantic graph encodes only the constraints that the user has expressed so far, while the SparQL query required by the knowledge base needs to be very specific and complete with regard to all the information to be returned. For example, in Figure 7-2, the green part shows the original semantic graph generated by the dialog manager that is equivalent to the user request for "an animated movie". However, it says nothing about the theater where the movie is shown, or the date and time of the showing. The acquisition of this information is essential for BCDR to successfully plan for the movie going activity. In order for the two components to talk to each other, we need to bridge these two discrepancies.

Two auxiliary components are added between the dialog manager and the knowledge base to address these issues. Based on the activity information from the dialog manager's activity library, a simple query reasoner expands the original semantic graph with new query target nodes representing any missing information to be retrieved from the knowledge base in order to completely ground the requested events. It is also responsible for filling in default information such as date, time, location if user does not specify them. After the original semantic graph is augmented, a FOL

to SparQL translator then maps the FOL predicates that encode various constraints into SparQL relations so that a valid knowledge base query can be generated.

When the knowledge base returns a list of results that meet all the constraints specified in the query, the dialog manager in turn needs to integrate these grounded event instances into its semantic graph. Figure 7-2 shows an example of the expanded semantic graph that integrated the two ground instances of the showing of an animated movie. This expanded semantic graph is also the base for which a TPN is generated so that BCDR can step in and find a valid activity plan for the user. This integration step is discussed in detail in the next section.

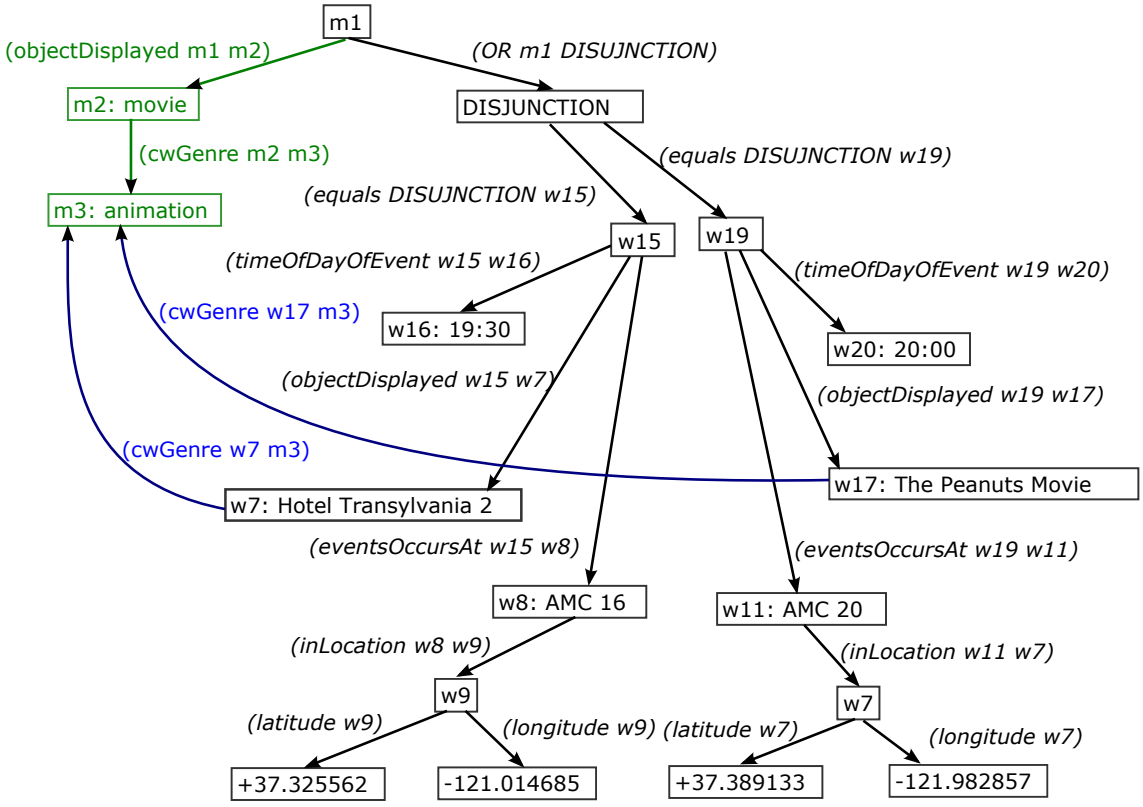


Figure 7-2: Semantic graph with grounded activities

7.1.2 Dialog Manager – BCDR Interface

The dialog manager passes TPNs to BCDR after grounding the activities with options returned from the knowledge base. BCDR then evaluates different choices in the

TPNs, and generates plans and relaxations that best meet the users' requirements. The output from the dialog manager is a set of first order logic expressions encoded in a semantic graph, while the input to BCDR is a set of activities and temporal constraints encoded as a TPN. We need to find the mapping between them such that (1) each activity's temporal and spatial requirements can be extracted from the semantic graph and encoded in the TPN; and (2) the choices and relaxations in BCDR's output can be mapped back to the nodes in the semantic graph, such that the dialog manager can present them to the user.

Input: The root of a branch in the semantic graph: *Root*.
a TPN $\langle P, Q, V, E, L_e, L_p, f_p \rangle$.
Output: An episode-graph node map $Map\langle E, N \rangle$.

```
1  M ← {};
2  p ←CreateVariable(Root);
3  disjunctionNode ←GetChildNode(Root);
4  for choiceEdge in GetOutEdges(disjunctionNode) do
5  |  optionNode ←toNode(choiceEdge);
6  |  q ←CreateAssignment(optionNode, p);
7  |  Queue ← {optionNode};
8  |  while Queue ≠ ∅ do
9  |  |  node ← Dequeue(Queue);
10 |  |  for edge in GetOutEdges(node) do
11 |  |  |  label ←Label(edge);
12 |  |  |  toNode ←toNode(edge);
13 |  |  |  if label ==timeOfDayOfEvent then
14 |  |  |  |  GetTime(toNode, E, q, L_e, M);
15 |  |  |  else if label ==eventsOccursAt then
16 |  |  |  |  GetLocation(toNode, E, q, L_e, M);
17 |  |  |  else if label ==InLocation then
18 |  |  |  |  GetPosition(toNode, E, q, L_e, M);
19 |  |  |  else if label ==objectDisplayed then
20 |  |  |  |  GetEventName(toNode, E, q, L_e, M);
21 |  |  |  else
22 |  |  |  |  Queue ←Queue ∪ toNode;
23 |  |  |  endif
24 |  |  end
25 |  end
26 |  Q ← Q ∪ {q}
27 end
28 P ← P ∪ {p}; return M;
```

Algorithm 17: Breadth-first exploration of semantic graph

We take a breadth-first approach to explore the branch in a semantic graph for an activity, and extract any temporal and spatial information about it. The pseudo code is presented in Algorithm 17. The algorithm starts from the root of the branch, such as the *m1* node in Figure 7-2, which represents the activity specified by the users. It then iterates through all child nodes of the root to search for any grounded candidates returned from the knowledge base, which are connected to the root node through a $DISJUNCTION$ node (Line 3 and 4). We create an activity variable for the root (Line 2), and the child nodes under $DISJUNCTION$, which are grounded options for the activity, are encoded as domain assignments (Line 5). Next, we explore the sub-branches under each of the child node to extract details about the grounded option (Line 7). Every time the algorithm sees an edge with certain labels, such as *eventOccursAt* and *inlocation*, it will initiate a corresponding function to extract the data (Line 12-18). The data is then associated with the episode for the candidate, either as name, location, or temporal constraints. Note that during the encoding process of a TPN from a semantic graph, a map is created from each TPN element to the corresponding semantic graph node (M, Line 13-19). It is stored and used by the dialog manager to interpret BCDR's solutions later.

For example, Figure 7-3 shows a branch of the semantic graph for a movie activity. This branch encodes two movie showing that meets the description *animation movie*. They are in different theaters, and starts at different times. The equivalent TPN, shown at the bottom of the figure, encodes the two alternative options. Each of the movie showing is modeled by an episode, with theaters and their locations. In addition, the episodes connect to the start event of each episode encodes the start time of the movie showing, relative to the trip start time (6pm). In addition, we keep a record of the mapping between the nodes in the branch and the constraints in the TPN. This is represented by the node IDs tagged to the episodes' names, locations, and durations. Once a solution is generated, we can use the mapping to convert BCDR's choices, such as selecting the episode with $w11, w12, w17$, to a natural language expression, *'You can watch The Peanuts Movie at AMC 20 for the movie activity'*.

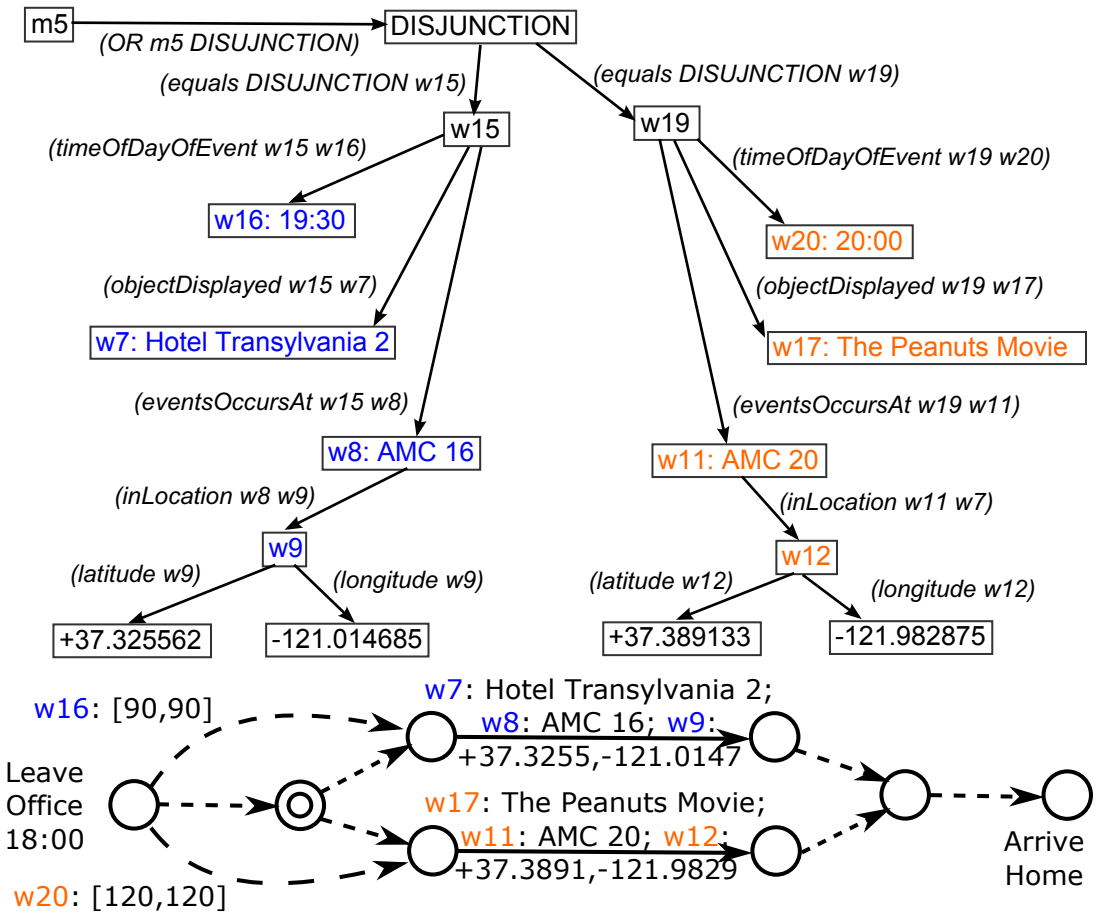


Figure 7-3: The TPN generated from a semantic graph branch

Currently, this Dialog Manager-BCDR interface can handle constraints related to the position, duration, start, and end times of an activity. The timing information is used by BCDR to evaluate temporal feasibility, while the positions are used by its routing function to estimate travel times between locations. As demonstrated in the example, each candidate option for an activity is mapped to a unique assignment and a set of episodes in the TPN. There are a few types of constraints over discrete variables that cannot be handled by the interface, such as the sequence requirements between activities. They are being processed separately using a set of predefined templates and added to TPNs directly.

7.2 Chapter Summary

In this chapter, we introduced the design and implementation of Uhura, an advisory system that can help users plan by integrating a dialog manager, BCDR and a knowledge base. As part of future work, we are actively working to expand Uhura's range of applications, and address the limitations that surfaced in the user study (presented in Chapter 8). These include better use of user preferences and common sense reasoning in Uhura's plan elicitation and relaxation process, which should provide higher quality solutions for the users.

Chapter 8

Applications and Empirical Evaluations

In this chapter, we discuss the four applications of BCDR with continuous and domain relaxation extensions, in the domains of deep-sea mission planning, transit vehicle scheduling, resource-constrained project scheduling and urban travel planning. Within each of the applications, we present the modeling of the problems, the configurations of BCDR for solving them, and the experiments for evaluating BCDR's performance compared to alternative approaches. These problems are of very different structure and scale, and through these experiments, we explore the strength and weakness of BCDR.

All experiments presented in this section were conducted on a desktop computer with an Intel Core i5-3470 processor and 16GB of RAM. The SNOPT optimizer we used is version 7.2, and the Gurobi optimizer is version 6.5.1.

8.1 Managing Deep-sea Exploration Missions

As presented in the beginning of this thesis, BCDR has been incorporated as part of a mission advisory system for helping ocean scientists schedule autonomous underwater vehicle tasks in deep-sea expeditions. Their missions are usually weeks long, and involve the operations of several AUVs. Each vehicle usually performs ten to fifteen

dives in a mission, and a dive may last between 6 to 16 hours. During each dive, a vehicle is tasked with a set of survey locations on the sea floor: the vehicle needs to traverse between each location, take samples and images, and return before running out of power. Due to unexpected ocean currents and incomplete terrain data, the traversal time between survey sites is highly uncertain and difficult to estimate. It is almost impossible for the scientists to correctly assess the uncertainty of each dive and plan tasks accordingly to meet the risk bound and a large set of operational constraints.

The advisor can check the feasibility of a mission plan and search for valid risk allocations that meet the risk requirement. If no such allocation can be found, the decision aid will explain the cause of failure using the conflicts detected during the search, and propose preferred relaxations to resolve the over-subscribed plan. If the users are not satisfied with the results, they can ask the mission advisor to adjust the solutions given their new inputs.

We examined the performance of BCDR on problems generated from this domain, with different sizes and complexity. The test cases were created using a mission simulator for underwater expeditions. Given a set of target locations on a map, the simulator generates survey tasks around them and connects these locations with traversal activities. Each test case describes the operations of multiple AUVs over several dives, and each vehicle's dive may contain multiple survey tasks. In addition, the traversals are represented by probabilistic durations, while the survey times and battery restrictions are modeled by simple temporal constraints. The operational risk limit is specified by the chance constraints in the cc-pTPNs. Multiple underwater robots may be deployed and working in parallel during a dive: each robot may have different speed and power storage. Depending on the distance and vehicles, probabilistic durations of different traversals and robots have different distributions.

These test problems have a very unique *relay* type structure: resources are being used repeatedly for different tasks, with constraints restricting the start times, end times or durations of the tasks. In addition to deep-sea exploration, this structure is also shared by problems in many other domains, such as scheduling vehicle usage in

a car-sharing network.

8.1.1 Setup

The randomly generated mission cc-pTPNs have a similar structure to the example presented in Chapter 2. To make it more complex, we extended the problems to include multiple vehicles and dives: there is always another scientist waiting for the shared vehicle following each dive; and there are multiple vehicles that are operating in parallel. We use the following control parameters in the mission problem generator to characterize the complexity of a test case:

- N_d: number of dives per vehicle. $1 \leq N_d \leq 5$.
- N_r: number of autonomous underwater vehicles available. $1 \leq N_r \leq 12$.
- N_{act}: number of activities per dive. $1 \leq N_{act} \leq 4$.
- N_{opt}: number of alternatives per activity. $2 \leq N_{opt} \leq 6$.

The total number of discrete variables in a test case is $N_d \times N_r \times N_{act}$, and the domain size of each variable is determined by its N_{opt}. Each problem is constructed as follows. We randomly sample locations from a region in Northern Pacific, within a 10km radius of (33.251, -121.555). The traversal times are computed using an average speed randomly selected between 10 and 20 kilometers per hour. The survey length at each location and the dive duration, T_{act} and T_{div}, are randomly sampled in $[10, 90]$ and $[60, 960]$ (minutes), respectively. These durations are encoded as episodes with relaxable temporal bounds. We define linear preference functions over these relaxable episodes with gradient (cost per minute) sampled between 0 and 10. The reward for each variable assignment, denoting a location selection for each survey activity, ranges from 0 to 1000. For example, Figure 8-1 presents an overview for the structure of a test case with 2 vehicles and each carries our two activities during their dives.

In total, we created 2400 test cases using randomly generated numbers of vehicles, risk bounds, survey locations and mission length. For each test case, we run BCDR with the following five configurations:

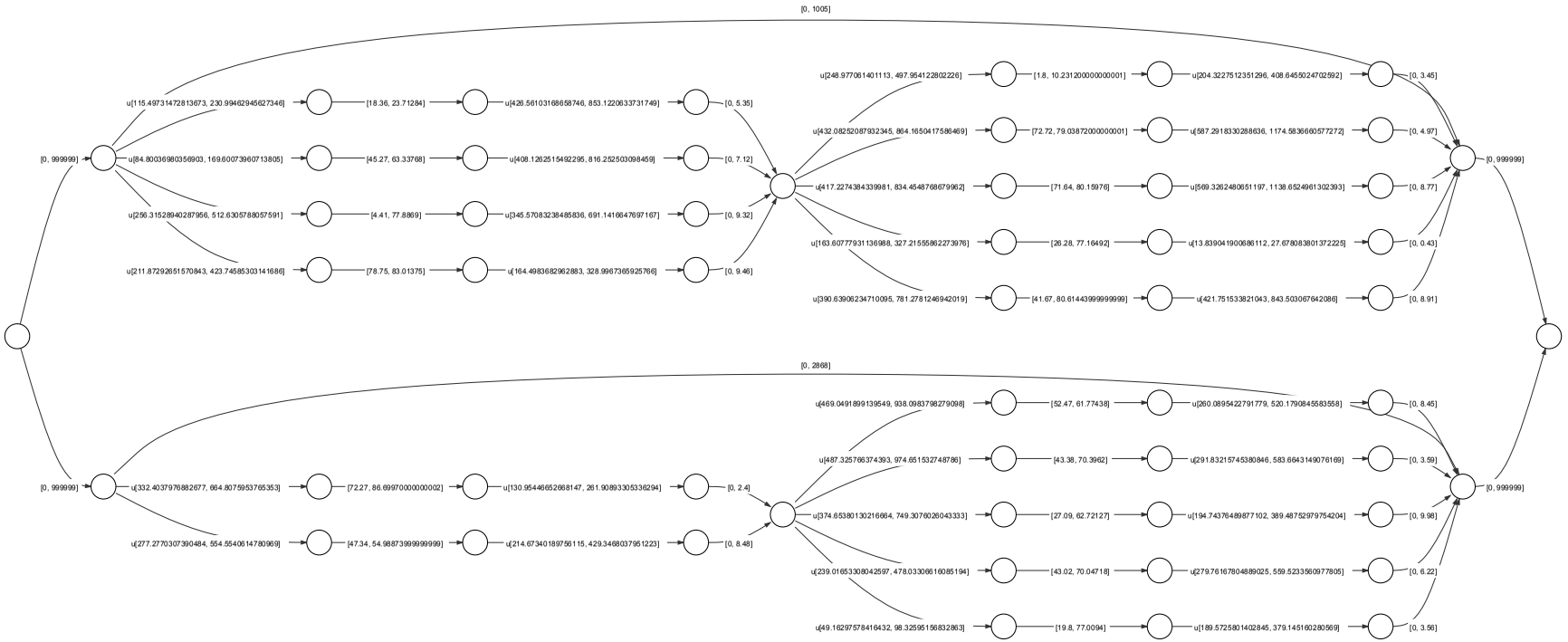


Figure 8-1: Overview of the structure for a test case

A. Consistency: determine and restore the feasibility of the test cases using BCDR. This algorithm is denoted as 'BCDR'. The cc-pTPNs are treated as TPNs by this configuration: the uncertain durations are assigned lower and upper bounds equal to their mean $[\mu, \mu]$.

B. Strong Controllability: using BCDR-U with a strong controllability model to determine and restore the feasibility of the test cases. This algorithm is denoted as 'BCDR-U(SC)'. The cc-pTPNs are treated as TPNUs: the uncertain durations are assigned lower and upper bounds computed from their mean and variance $[\mu - 3\sigma, \mu + 3\sigma]$.

C. Dynamic Controllability: using BCDR-U with a dynamic controllability model to determine and restore the feasibility of the test cases. This algorithm is denoted as 'BCDR-U(DC)'. The cc-pTPNs are treated as TPNUs: the uncertain durations are assigned lower and upper bounds computed from their mean and variance $[\mu - 3\sigma, \mu + 3\sigma]$.

D. Chance-constrained Strong Controllability: using BCDR-C to find a grounded TPNU of the cc-pTPN that is strongly controllable while meeting the chance constraint, or a set of relaxations for the cc-pTPN that will enable such a TPNU. This algorithm is denoted as 'BCDR-C(SC)'.

E. Chance-constrained Dynamic Controllability: using BCDR-C to find a grounded TPNU of the cc-pTPN that is dynamically controllable while meeting the chance constraint, or a set of relaxations for the cc-pTPN that will enable such a TPNU. This algorithm is denoted as 'BCDR-C(DC)'.

8.1.2 Results

In this experiment, we benchmarked BCDR and its variants on each problem using the five aforementioned configurations. We use SNOPT as the linear optimizer for consistency and controllability based tests, and non-linear optimizer for chance-constrained tests with probabilistic durations. In each test run, the time consumption

until the first solution returned, numbers of conflicts detected, as well as the utility of the solution, were recorded. The timeout for each run was set to be 30 seconds, which is usually the maximum duration scientists are willing to wait for.

First, we present the results for Configuration A, B and C, which are BCDR, BCDR-U(SC) and BCDR-U(DC). The runtime performance of the three algorithms are presented in Figure 8-2. Each dot in the graphs represents the cumulative number of instances solved on problems that contains equal or less number of constraints, which are indicated by the x-axis. In total, BCDR with temporal consistency checker solves 1589 problems within 30 seconds, while the number for BCDR-U(SC) and BCDR-U(DC) are only 430 and 695, respectively. As can be seen from the figures, BCDR with consistency assumption is able to solve problems with up to 1000 constraints, while the other two algorithms never succeeded with problems beyond 500 constraints. This is the result of the different handling of uncertain durations: due to the consideration of all possible outcomes from each uncertain duration, both controllability-based BCDR-U algorithms are more restrictive compared to the consistency-based BCDR, which treats uncertain durations as controllable and may only satisfy one of its many outcomes. Hence the number of conflicts detected and resolved by both BCDR-U(SC) and BCDR-U(DC) are much higher than that of BCDR (Figure 8-3, 8-4 and 8-5), which significantly impacts their performance on large problems.

In addition, checking strong and dynamic controllability are also more expensive operations than checking consistency, which contributed to their lower runtime performance. As presented in Chapter 4, we use the Bellman-Ford algorithm for consistency checking and negative loop extraction ($O(N^2 logN)$). We add another layer of triangular reduction ($O(N^2)$) on top of it for checking strong controllability. For dynamic controllability, we use the FASTDCCHECK algorithm $O(N^4)$ from (Morris, 2006). It is interesting to observe that BCDR-U(DC) out-performs BCDR-U(SC), as there is an order-of-magnitude difference in their checking algorithms' runtime complexity. This is likely the result of the smaller amount of conflicts that were resolved by the dynamic controllability version, as shown in Figure 8-4 and 8-5. Dynamic controllability allows

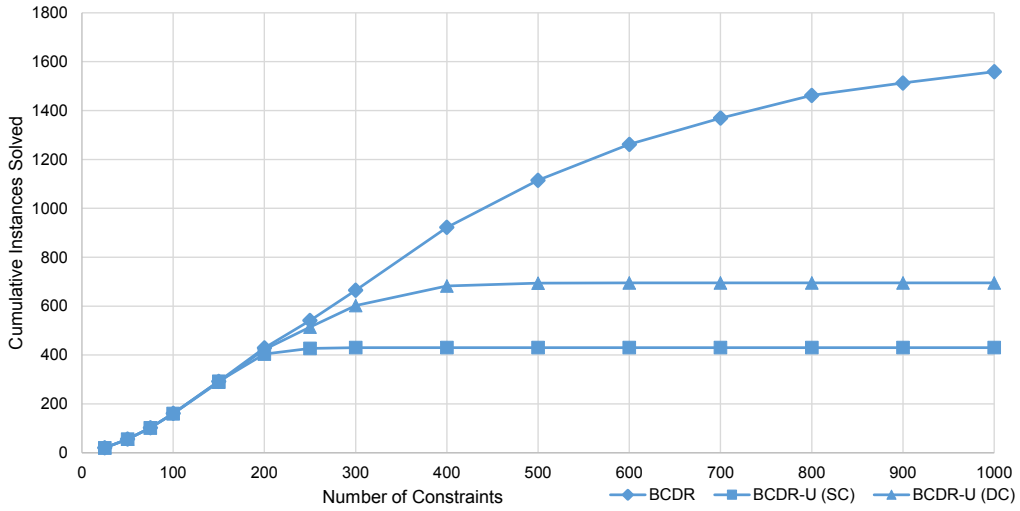


Figure 8-2: Cumulative number of instances solved by BCDR (in 30 seconds)

more flexibility than strong controllability in that it does not require a static schedule to accommodate all constraints and uncertain durations. The runtime of BCDR-U is dominated by conflict resolution for over-subscribed plans, hence the less time spent on conflict resolution compensated for the additional time on controllability checking.

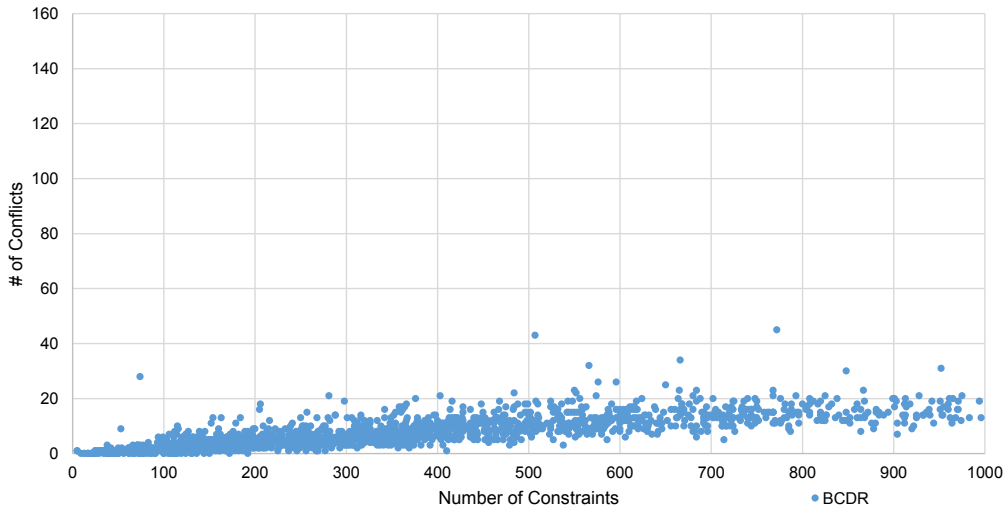


Figure 8-3: Conflicts detected by BCDR (Consistency)

Next, we discuss the results of BCDR-C(SC) and BCDR-C(DC) (Configuration D and E), the two chance-constrained relaxation algorithms with strong and dynamic controllability checker. They were used to restore the feasibility of cc-pTPNs through both temporal and chance constraints relaxations. The results is shown in Figure 8-6.

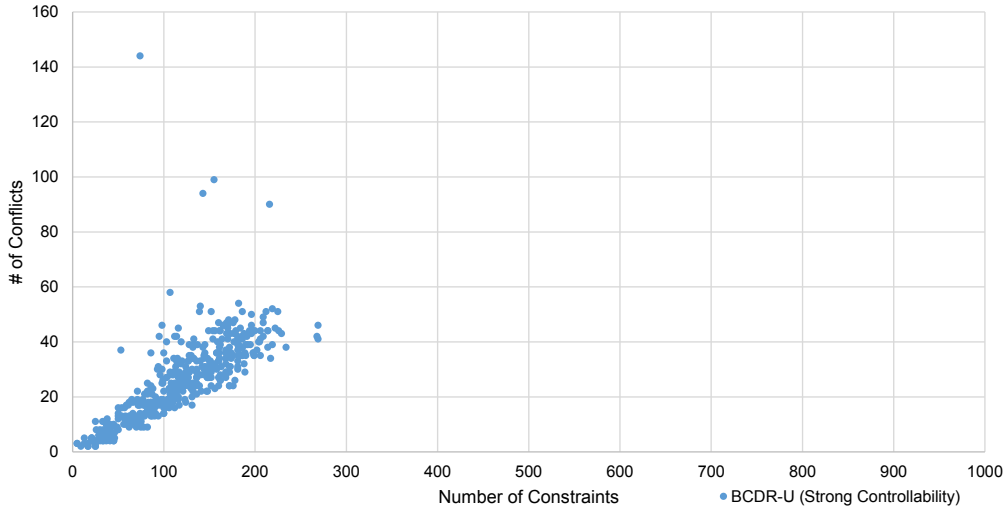


Figure 8-4: Conflicts detected by BCDR-U(SC)

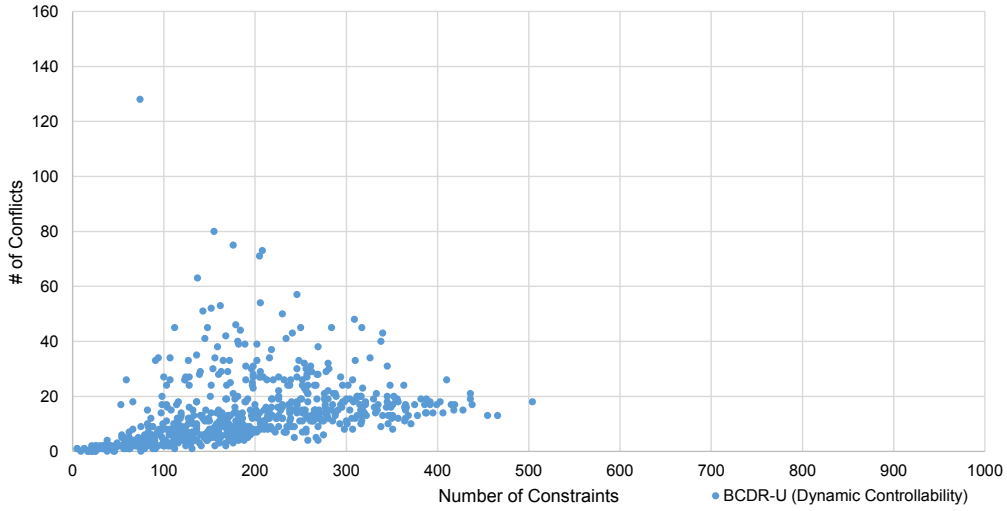


Figure 8-5: Conflicts detected by BCDR-U(DC)

In total, 269 of 2400 tests were solved by BCDR-C(SC) in 30 seconds, while the number for BCDR-C(DC) is 328 of 2400. Similar to the BCDR-U experiment, the dynamic controllability version performs better than the strong controllability version due to the smaller number of conflicts. BCDR-C solves fewer problems than BCDR-U and BCDR within the same time limit. This is mainly due to BCDR-C's chance-constrained conflict resolution procedure: it adds a non-linear constraint for risk allocation to the optimiation problem, which makes it significantly harder to solve than the linear optimization problem required by BCDR and BCDR-U. As

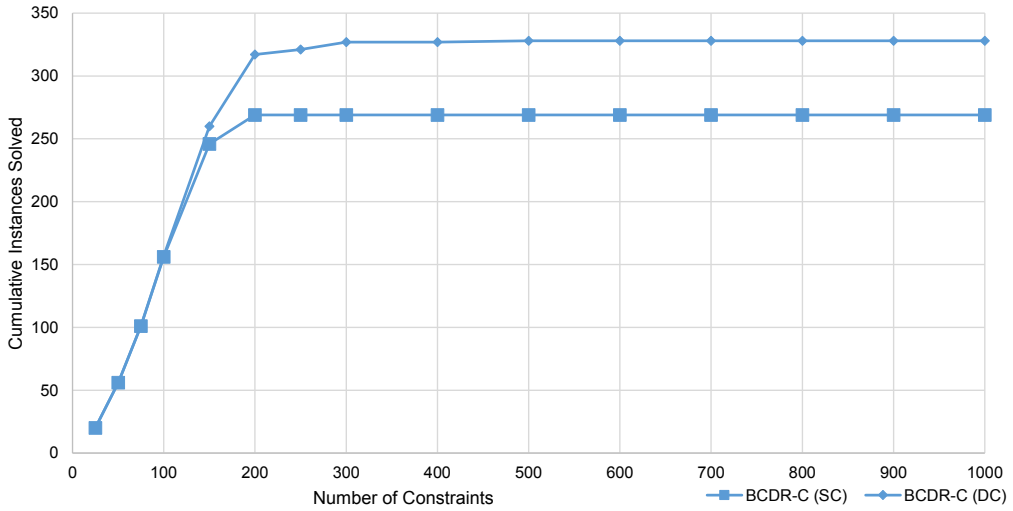


Figure 8-6: Cumulative number of instances solved by BCDR-C (in 30 seconds)

can be seen in Figure 8-7 and 8-8, the number of conflicts detected by BCDR-C are not larger than those detected by BCDR-U (Figures 8-4 and 8-5). Solving the optimization problems for conflict resolution is the most expensive operation in the BCDR algorithm, which takes up to 90% of the total computation time. Hence the extra computation was mainly due to the expensive non-linear optimization process.

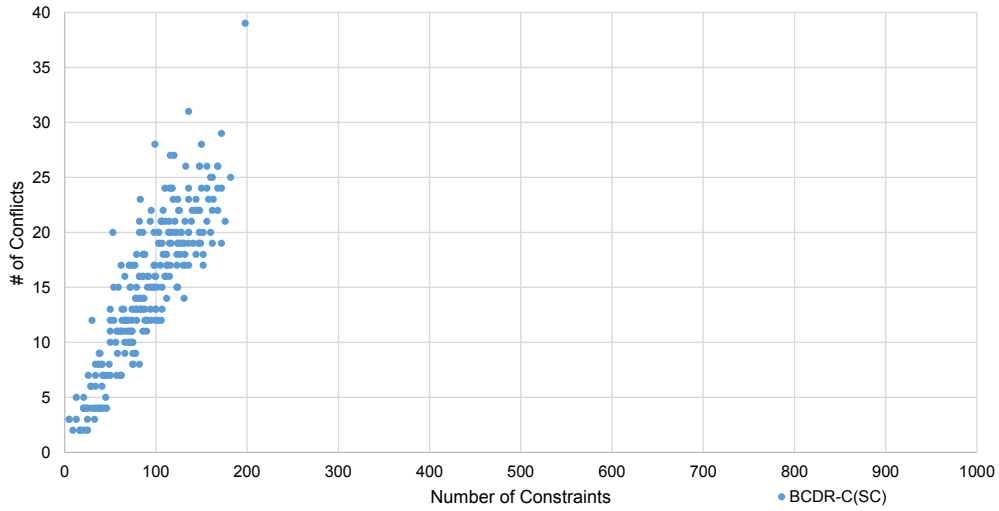


Figure 8-7: Conflicts detected by BCDR-C(SC)

BCDR performs much better on problems with set-bounded uncertainty models, since all constraints are linear during conflict resolution. On the other hand, non-linear models, such as a normal distribution, is better for describing the uncertainty

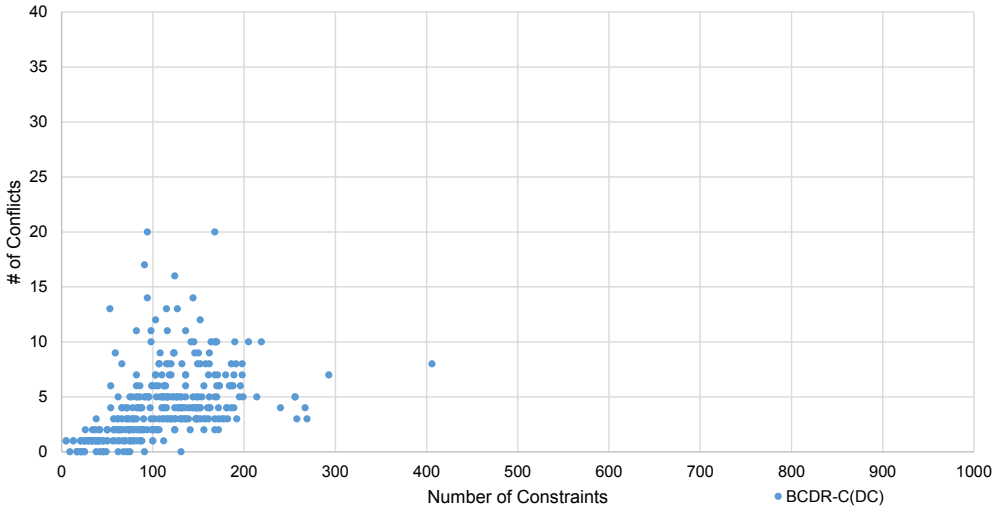


Figure 8-8: Conflicts detected by BCDR-C(DC)

in some real world activities (e.g. the timing of natural phenomena). Even though it requires much more computation time, BCDR-C was able to resolve most of the problems with less than 200 constraints in 30 seconds. This is enough for modeling a 10-hour survey mission of several underwater vehicles working in parallel. In general, the algorithm configuration and modeling should be different from one application to another. The choice of uncertainty and controllability model to use is thus application dependent. In addition, results from this experiment also suggest that tractability must be considered when selecting these models.

8.2 Optimizing Dispatching Strategies for Maintaining Headways on Transit Routes

In this section, we discuss a different application of BCDR in the domain of transit system management. Due to unexpected delay in travel and dwelling, transit vehicles sometimes cannot operate on schedule and maintain their designated headways. If a vehicle is delayed and operating off schedule, the gap between it and an earlier vehicle will increase, causing it to carry more passengers, spend more time at each station for dwelling and get delayed even more. In some extreme scenarios, passengers waiting at a station may see two or more vehicles along the same route arrive together, and

find an overcrowded vehicle followed by near-empty ones. This problem is often called *bunching* or *platooning* in public transportation, and is a major challenge for reliable transit service.

This problem has been investigated by many researchers in operation research, and more details can be found in (Bellei & Gkoumas, 2010). Here we present a simple example, constructed based on an SBS Transit article on bus bunching[1], to explain this problem. Assume that three three vehicles are running in 8-minute intervals and demand along the route is constant (Figure 8-9).

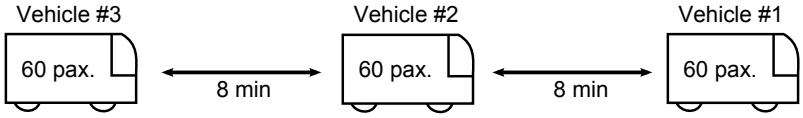


Figure 8-9: Regular headways between vehicles

Vehicle #1 and #3 did not experience any delay in its operation and was able to keep to schedule. Vehicle #2 encountered some issues while dwelling at a previous station. Hence, the headway between Vehicle #1 and #2 was lengthened, while the headway between Vehicle #2 and #3 was shortened (Figure 8-10).

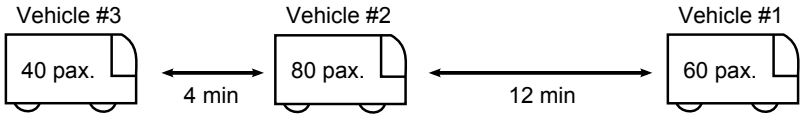


Figure 8-10: Uneven headways and passenger load due to delayed Vehicle #2

By taking more than its share of passengers Vehicle #2 slowed down as a result, while Vehicle #3 picked up fewer passengers. As this continues, Vehicle #2 would eventually bunch up with Vehicle #3. To a passenger waiting for Vehicle #2, he would have waited for 6 minutes, and it would seem that Vehicle #2 was crowded while Vehicle #3 was relatively empty.

Current approaches for preventing vehicle bunching heavily rely on the operators' experience and intuition: they have to respond quickly enough to any irregular operations before they propagate to the entire route. In addition, the operators have a

[1] 'Sometimes, two buses of the same service arrive at the same time with one bus being overcrowded and the other almost empty. Why?', `http://www.sbstransit.com.sg/doyouknow/facts_bus.aspx`.

very limited set of actions to take, such as asking the delayed buses to skip stations or urge the passengers to wait for the next bus. Both may cause inconvenience for passengers either on board or waiting at stations.

Here, we present a dynamic scheduling approach for managing a transit route to address this problem. Given the schedule of an existing transit route and historical performance data, we use BCDR to compute a robust dispatching strategy for the vehicles on this route, such that the headways between buses/trains can be better maintained. It builds in additional buffer time for vehicles to wait at each station based on the uncertainty, which allows a dynamically controllable dispatching strategy to be generated. The dispatching strategy, represented as an execution policy for events in the scheduling problem, provides real-time guidance for pausing vehicles at each station in order to maintain headways along the route. For the passengers, it means that the frequency of services is more regular, such that they are less likely to wait for an extended period of time for the service, or board an overly crowded vehicle.

Our approach is similar to the frequency-based method proposed in (Bartholdi & Eisenstein, 2012), which has each bus observe the preceding and following ones, and strategically delay themselves at stations to maintain regular headways. As presented in the same paper, this method has been shown to outperform prior work in controlling the university bus system at Georgia Institute of Technology. The major difference between it and our scheduling based approach is that we used a centralized algorithm and pre-compute the strategy based on historical data. It allows us to coordinate all vehicles along the route simultaneously, and restore from service interruptions quickly without waiting for the bus delays to propagate from one to another. However, the trade-off is that the dynamic policy requires more computation time to generate, and the policy itself may become invalid if any travel or dwell time along the route falls outside the set-bounds for uncertain durations.

The objective of this experiment is to explore different problems that BCDR with the continuous relaxation extension can solve and benchmark its scalability. The results have not been compared to other approaches in the field. We have not verified

this approach on any real transit routes, though we would very much like to share it and evaluate this approach in a real-world system.

8.2.1 Setup

We selected the Red Line subway [2] in Boston for this experiment. It is the busiest mass transit route in the city, carrying more than 250,000 passengers per day. The agency operating Red Line, the Massachusetts Bay Transit Authority, has been publishing performance data for every train operation since June 30, 2015. Using their API, we were able to retrieve the travel times of each train between stops, and the amount of time the train dwelled at every stop. Combined with the published schedule, we constructed a TPNU for modeling the operation of the Red Line. To simplify the problem, our model covers only the inbound direction trains from Alewife station to JFK/UMass station (before the line branches into two directions).

The travel time of each train between stations and dwell times are represented by uncertain durations, while the scheduled headways are encoded as episodes with controllable temporal constraints. Figure 8-11 is cropped from the visualization of the problem, which shows a small portion of the TPNU that presents all basic elements: dwell, wait for departure, and traversal to the next station. Given a train ride, the three elements are repeated for each station pair. The collection of constraints for one train ride are then repeated for all 167 trains during a peak day along the inbound direction. Between neighboring trains, the headway constraints are added between their their arrival events at a station. The temporal bounds of these constraints are defined using the scheduled headway: $[Headway - 1, Headway + 1]$. We slightly weakened the headway requirements from the schedule by ± 1 minutes to allow some flexibility for handling uncertainty, such that more efficient solutions can be generated.

The bounds for the uncertain durations, travel times, and dwell times are estimated using the historical performance data: given a train ride between station A and B, we retrieve train operation data between the same stations in history and compute a lower and upper bound to cover them. For example, for a train ride that

[2] `http://www.mbta.com/schedules_and_maps/subway/`

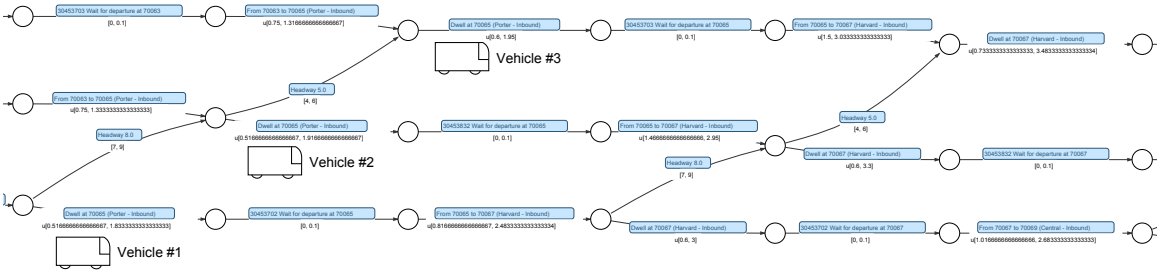


Figure 8-11: Basic elements in the TPNU for Red Line trains

leaves station Alewife at 5pm, we will retrieve all records for previous trains that left between 4:30pm to 5:30pm during weekdays, and collect their travel times between stations and dwell times. For its uncertain duration, the lower bound is defined using the smallest travel time in the collection, while the upper bound is chosen such that 98% of all data points are covered between the lower and upper bounds.

Given the uncertain durations, there is no execution policy that meets all constraints in the TPNU. The objective of this experiment is to restore the dynamic controllability of the over-subscribed TPNU, by building in the minimal amount of wait times at each station. We can then compute a dispatching strategy that is robust against all uncertain travel and dwell times can be found from the relaxed TPNU. Therefore, the only relaxable elements in the TPNU are the upper bounds of the wait times at each station. The values of the upper bounds are initialized to be 0.01 minute, and associated with a linear cost function with gradient 1 to penalize any excessive delays.

8.2.2 Results

In this experiment, we solved the over-subscribed plans using two approaches: conflict-directed relaxation with BCDR-U(DC), and a MIP encoding with Gurobi. We selected Gurobi instead of SNOPT for this experiment due to the integer variables in the MIP encoding. For BCDR-U(DC), we use Gurobi as its sub-solver to compute optimal relaxations for learned conflicts. For the MIP encoding approach, we use Gurobi directly to solve the complete problem. The MIP encoding we used was first introduced in (Wah & Xin, 2007) for checking dynamic controllability, and later

modified by (Cui et al., 2015) to evaluate the robustness of schedules for resource-constrained project scheduling problems. The objective function for both approaches are set to find the minimum cost relaxations, which correspond to the minimum delays that have to be built into each station, and makes the TPNU for the transit route a dynamically controllable network.

Different from the AUV mission planning problems, the transit operation problem's temporal network is highly connected due to the headway constraints. They create many more cycles in the network, which result in more conflicts between constraints. The larger number of conflicts makes the problems significantly more difficult to solve. Therefore, given the limited computing resources and experiment time, we do not expect both approaches to solve the complete problem with 169 trains and around 10,000 constraints. To benchmark and compare the performance of two approaches, we generated a set of smaller test problems by capturing only a subset of the trains and stations. Each of the 48 test problems contains N ($2 \leq N \leq 7$) trains and M ($2 \leq M \leq 9$) stops. In this experiment, we set the timeout of each test run to be 10 minutes.

7	182.62	x	x	x	x	x	x	x
6	87.051	x	x	x	x	x	x	x
5	27.012	314.55	x	x	x	x	x	x
4	7.1596	120.48	370.16	x	x	x	x	x
3	1.9395	18.362	57.251	162.20	424.35	x	x	x
2	0.62508	3.2241	9.7885	19.904	51.379	80.885	200.97	346.31
Trains / Stops	2	3	4	5	6	7	8	9

Table 8.1: Runtime of BCDR-U with Gurobi as sub-solver (in seconds)

The results are shown in Table 8.1 for BCDR-U(DC), and Table 8.2 for Gurobi with MIP encoding. Rows indicate the number of trains captured by the test problems, while columns indicate the number of stops. The run-times of each approach for solving the test problem are the averaged results from five test runs. Within the time limit, BCDR-U(DC) solved 20 out of 48 problems with up to 7 trains/2 stops and 2 trains/9 stops, while Gurobi only solved 5 problems with less than 3 trains and

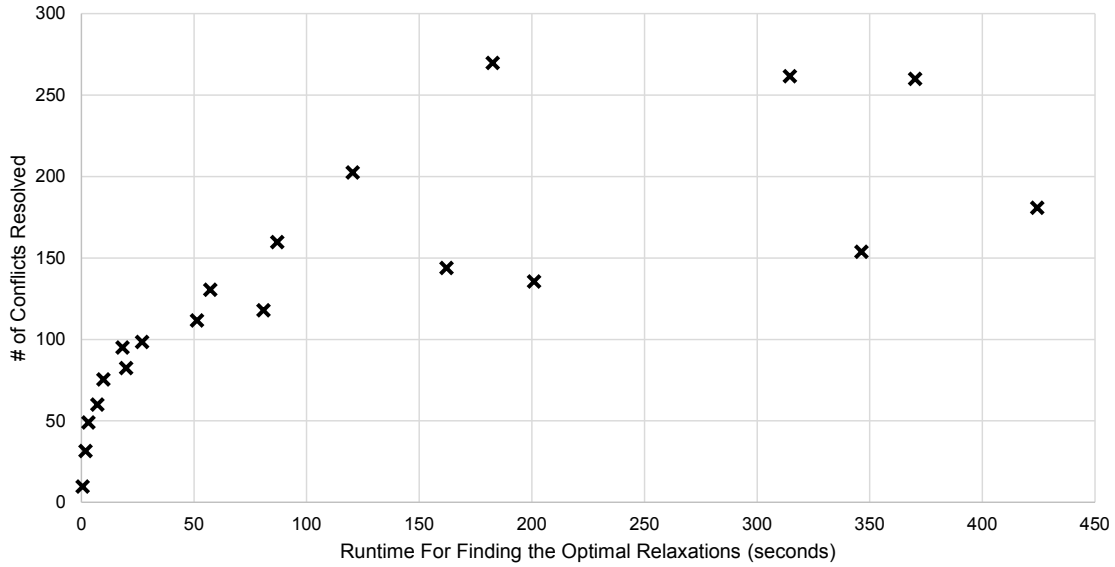


Figure 8-12: Number of conflicts resolved by BCDR-U before finding the optimal relaxations

7	x	x	x	x	x	x	x	x
6	x	x	x	x	x	x	x	x
5	x	x	x	x	x	x	x	x
4	x	x	x	x	x	x	x	x
3	30.399	x	x	x	x	x	x	x
2	2.7040	12.569	76.568	340.29	x	x	x	x
Trains / Stops	2	3	4	5	6	7	8	9

Table 8.2: Runtimes of Gurobi with MIP encoding (in seconds)

5 stops. For the problems that both methods solved in the time limit (2 Trains with 2, 3, 4, and 5 stops, and 3 Trains with 2 stops), BCDR-U(DC) is roughly one order of magnitude faster than Gurobi with MIP encoding. Similar to the results from the first experiment, the runtime of BCDR-U(DC) on an over-subscribed plan is largely determined by the number of conflicts in it. As can be seen in Figure 8-12, problems that require longer runtime for BCDR-U(DC) to solve often contain more conflicts.

The results demonstrate the advantage of BCDR-U's conflict-directed approach: it allows the solver to only deal with the conflicts instead of the complete problem, which significantly reduces the number of constraints and variables it has to consider. Although BCDR-U's conflict learning step requires a significant amount of compu-

tation, overall the procedure is still worth the effort, especially for large problems like the transit route optimization. Even for Gurobi, which is commonly regarded as a state-of-the-art MIP and LP optimizer, BCDR-U is still able to significantly improve its performance on the relaxation problems when compared to the direct MIP encoding approach.

8.3 Robustness Analysis of Resource-constrained Project Schedules

Finally, we present the application of BCDR to evaluating the robustness of resource-constrained project schedules. First presented by (Cui et al., 2015), the objective of this experiment is very different from the previous two experiments: instead of over-subscribed temporal plans, BCDR is given a feasible plan and a partial-order schedule for it, and is asked to find the maximum uncertainty that can be built into the durations of some activities, while maintaining the controllability of the problem. It is like finding a configuration of temporal problems that pushes them to the boundary of feasibility.

A partial-order schedule (POS) consists of a set of time constraints between activities such that any realization 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 modeled as an STNU 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, our measure of robustness is defined as the maximum deviation (i.e., width of the contingent bound) on any activity at which the STNU is dynamically controllable. It can also be viewed as a guarantee on the minimal amount of uncertainty that a POS can handle without any rescheduling.

8.3.1 Setup

To compute the maximum deviation allowed in a POS, we will need to slightly modify the BCDR-U algorithm and problem formulation. First, we initialize the upper bound of all activity durations to $+\infty$, which is the maximum possible uncertainty we can build in. It makes the schedule over-constrained, and allows BCDR-U to start from here to iteratively discover conflicts and restore the controllability of the schedule. Second, we modify the objective function used by BCDR-U (Equation 5.5) to be defined over the range of uncertain durations (Equation 8.1). Instead of minimizing the costs of weakening relaxable constraints, this modification allows BCDR-U to maximize the minimum deviation among all activity durations.

$$\max \min_{ub'_i} (ub'_i - lb_i); \tag{8.1}$$

Figure 8-13 presents a simple example that demonstrates the modification and the expected output from BCDR-U. Given a feasible schedule over two activities (Figure 8-13a), we first increase the upper bounds of the activities to $+\infty$ (Figure 8-13b). The change effectively applies the maximum possible deviation to the schedule, but also makes it over-subscribed in most scenarios. Then we apply BCDR-U(DC) on the problem, asking it to extract conflicts introduced by the modification, resolve them by lowering the upper bound according to the new objective function, and restore the dynamic controllability of the problem (Figure 8-13c).

As test cases, we use 325 partial-order schedules for RCPSP/max problems (Kolisch & Padman, 2001) with 10–18 jobs[3]. The schedules are generated by a scheduler that optimises a measure of POS flexibility (Banerjee & Haslum, 2011). The TPNU representation of a schedule has a time point for the start and end of each activity, as described above. Hence, the number of events and episodes with with uncertain durations is determined by the number of jobs, but the number of episodes with controllable constraints varies from 50 to 300.

[3]Set J10 from PSPLIB (`http://www.om-db.wi.tum.de/psplib/`)

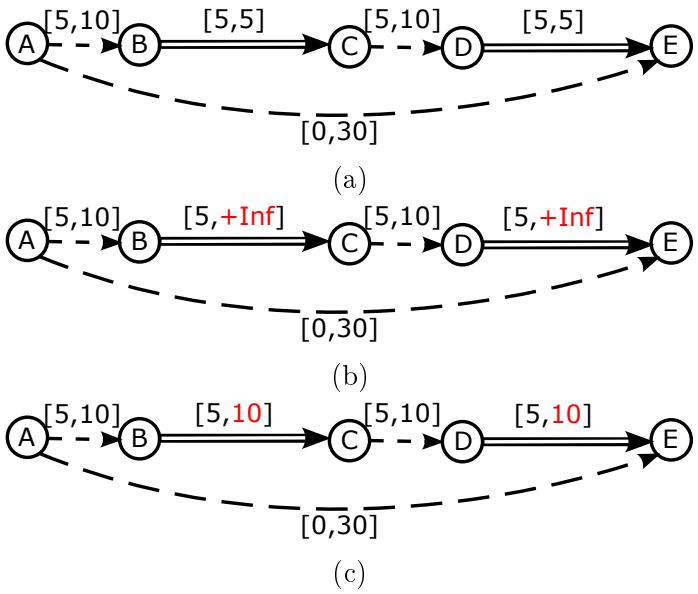


Figure 8-13: Examples for maximizing flexibility

8.3.2 Results

In this experiment, we solved the over-subscribed plans using two approaches: conflict-directed relaxation with BCDR-U(DC), and MIP encoding with Gurobi. Similar to the second experiment on transit route schedules, we use Gurobi as the sub-solver for BCDR-U(DC). For the MIP encoding approach, we used the encoding introduced in (Cui et al., 2015) to compute the maximum flexibility that can be built into the uncertain durations. The problems in this experiment are of much smaller scale compared to the previous two experiments, and both approaches were able to solve all problem within the time limit (30 seconds). For each test run, we recorded the computation times of both approaches for comparison.

The results are shown in Figure 8-14: each data point in the graph represents the averaged runtime for one problem over five test runs. Similar to the results from the previous experiments, BCDR-U(DC) is very effective for solving these problems when compared with the MIP/Gurobi approach: it is about one order of magnitude faster on solving most test problems.

Among the 325 test runs, we observed one interesting out-lier in the result: BCDR-U(DC) spends more time solving Instance 135 than Gurobi. This is the only case in

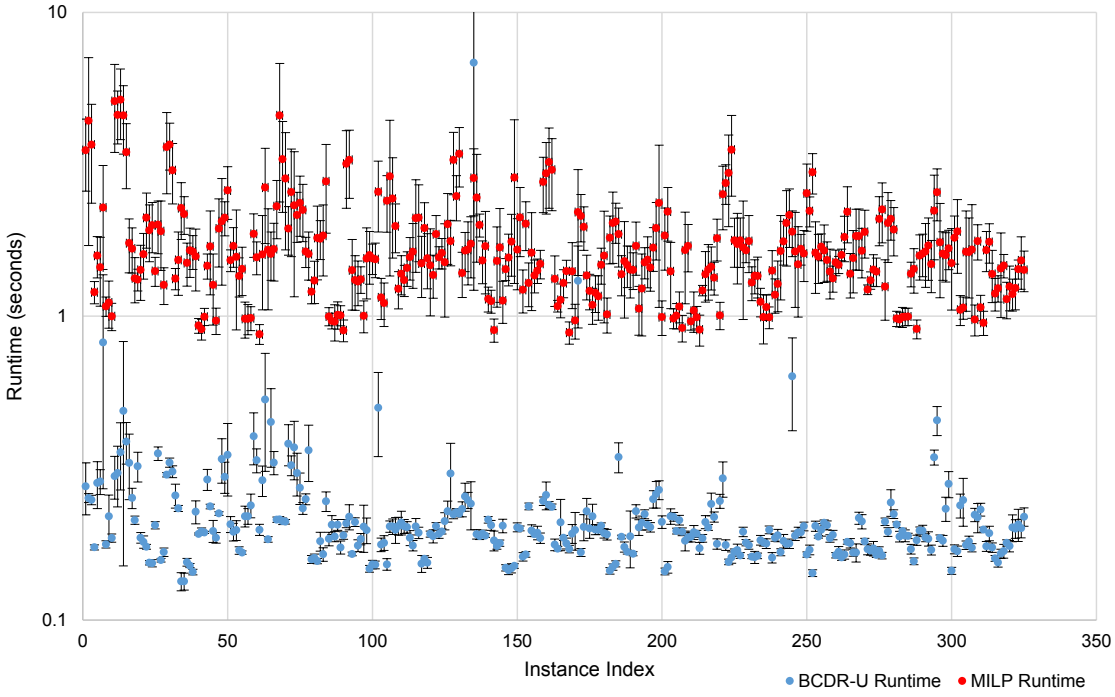


Figure 8-14: Runtimes of BCDR-U(DC) and Gurobi/MIP on RCPSP schedules (in seconds)

which BCDR-U(DC) spends more time, and we were interested in investigating the cause behind this issue. We started with the number of candidate relaxations evaluated in solving the problem (Figure 8-15). Not surprisingly, BCDR-U(DC) tested many more candidates before reaching the optimal relaxations for Instance 135 (the 825th candidate dequeued), which explains the longer runtime on this problem. Next, we retrieved the utility of candidates generated by BCDR-U(DC) while solving Instance 135. Figure 8-16 presents the utility of BCDR-U(DC)'s candidates in a test run: from the first feasible candidate (Index 1 with utility 7.6667) to the final solution (Index 825 with utility 5.1429). There are a few 'plateau' regions in the graph, which indicate that BCDR-U(DC) spent a lot of time evaluating candidates of similar utility without making progress towards the solution. This indicates one of the weakness in BCDR-U(DC)'s best-first enumeration approach: without a good heuristic function for computing the bounds on continuous relaxation costs, BCDR-U(DC) may waste a large amount of time exploring parts of the search space it believes to be promising, but which in fact contain no good solutions. This issue is more often observed

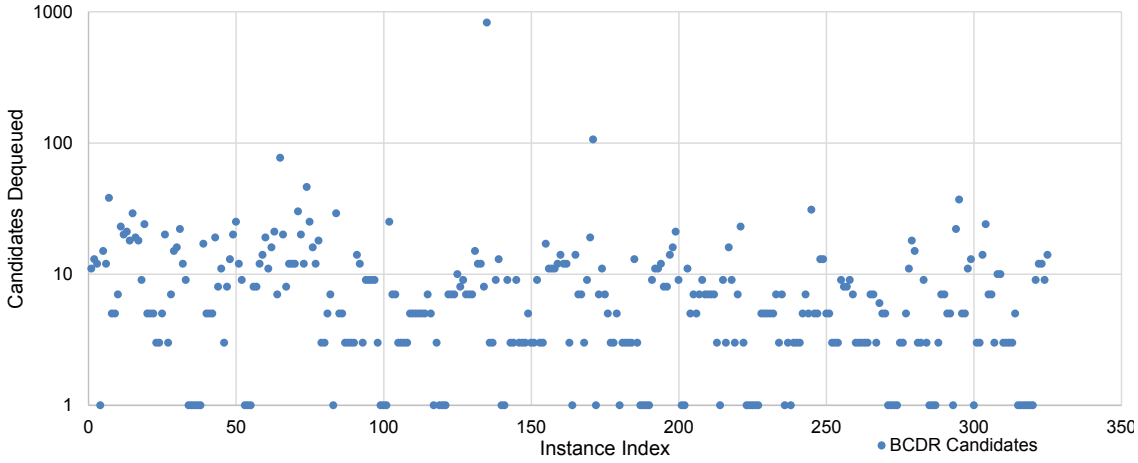


Figure 8-15: Numbers of candidates evaluated by BCDR-U(DC) on RCPSP schedule problems

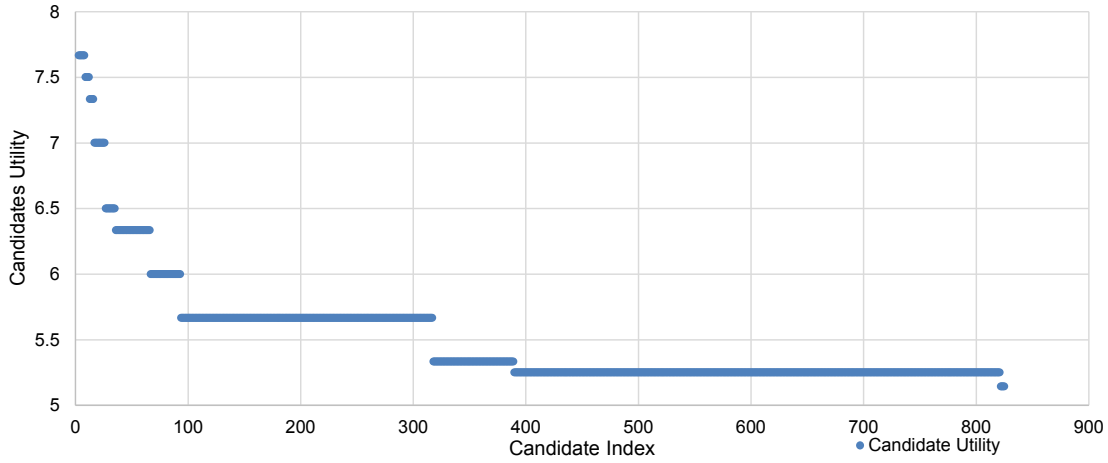


Figure 8-16: Utilities of candidates evaluated by BCDR-U(DC) while solving Instance 135

in problems with many conflicts, since each conflict may introduce multiple continuous relaxations and significantly enlarge the search space. Therefore, a heuristic function for estimating the bound on relaxation cost should be incorporated whenever available, and developing a general application bounding function for continuous relaxation is part of our future research.

8.4 Resolving Over-subscribed Travel Plans with Domain Relaxations

The objective of domain relaxation is to provide more options for users while resolving over-subscribed plans. As presented in Chapter 1, BCDR has been implemented as part of Uhura for users to manage their day-to-day tasks. To evaluate the usefulness of domain relaxation in such scenarios, we conducted a user survey using Uhura, which examines it in two aspects: (1) can it help users find solutions in scenarios that would be impossible to solve using only temporal relaxations, and (2) is the quality of BCDR's domain relaxations acceptable in different scenarios. In this section, we present the design of the user survey, and discuss the results and lessons learned.

8.4.1 Setup

The travel assistant behaves much like the example presented in Chapter 2. In the user survey we use a web-based GUI to interact with the participants, which provides step-by-step guidance for them. It operates on a set of template scenarios, and promotes the users to input their requirements and activities for their trips, such as origin, destination, time of departure, and desired trip length.

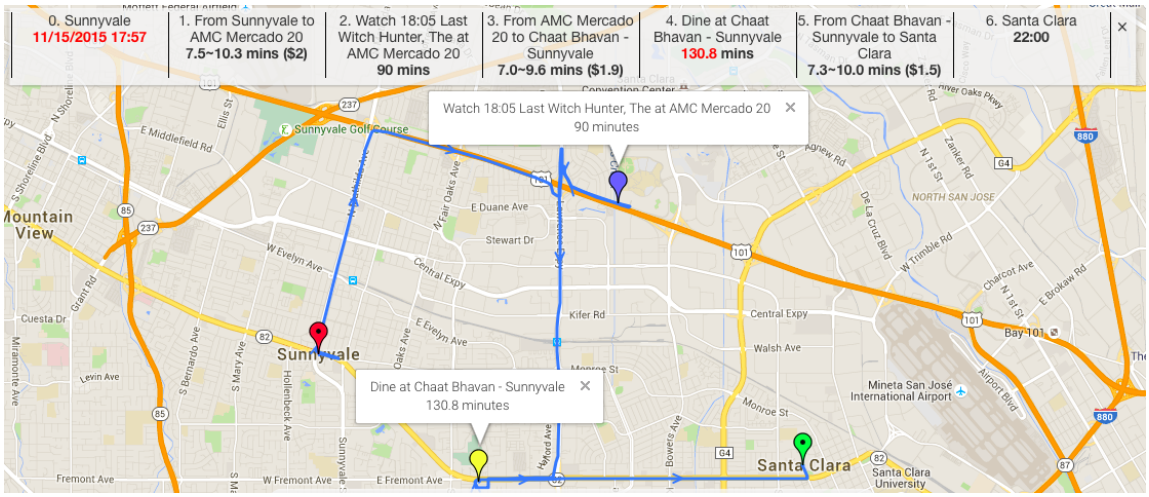


Figure 8-17: A trip plan presented in the web interface

Once a solution is bound, it will be sent back to the interface and presented to the

user, both using a story line, and visually using a polyline overlay on the map. For example, Figure 8-17 shows a trip plan with a dinner and movie, with relaxations to the departure time and dinner duration. The user is presented with an overview of the trip with the map visualization, on which the task locations and traversal routes are marked. In addition, the interface presents a story line of activities in this plan, as well as the time allocation over them. If any temporal requirements are relaxed by Uhura, they will be highlighted in the story line. Finally, if the user is not satisfied with a solution, they can send a *Next Solution* query along with the reason behind this decision to Uhura through the web interface. The reason could be to alternate a destination choice, or revert the relaxation to a constraint. Uhura will then search for the next best plan that respects these newly added requirements.

There are six scenarios in this survey, which are constructed based on commonly encountered travel planning problems, such as an evening outing, a date, or a weekend get together for kids. The users were asked to plan for up to three tasks in a session, which can be either lunch, dinner, or movie, subject to different departure and arrival time constraints. The six scenarios are defined as the following:

- *You are planning for an outing trip with friends after work. This trip may include a dinner and/or a movie. It starts from your office, and ends at your home.*

- *You are planning for a date. This date includes a nice dinner at a French restaurant and a movie. It starts from a meeting point you selected, and ends at your date's home.*

- *You are planning for a monthly weekend get together with a small group of close friends. This trip includes a 2-hour lunch, a movie and possibly a dinner. It starts from a meeting point you selected, and ends at your home.*

- *Your relatives are visiting today, and you are planning to take them out in the afternoon. This trip may include a lunch, a movie and possibly a dinner. It starts from your home, and ends at the airport they are flying out.*

- *You are planning to take parents out for dinner and movie this evening. They like Chinese food and drama. They also need go to bed early (before 9pm). This trip starts from your office, and ends at your home.*

- *You are planning to join a kids play date with lunch and a movie. Kids like spaghetti or tacos, and they only want animated movie. Note that you have to take them back by 3:30pm. This trip starts from a meeting point and ends at your home.*

Finally, at the end of each session, we asked the participants to evaluate the last solution proposed by BCDR, and submit two scores, quality and novelty, in the scale of 1 to 5. The quality score indicates if the user is satisfied with the solution, with 5 being very satisfied and 1 being not satisfied. The novelty score indicates if the plan produced by Uhura is something that is new or not thought of before by the participant, with 5 being very new and 1 being not novel at all.

8.4.2 Results and Discussion

We received survey results from nine different participants, for a total of 54 sessions. During the survey, we recorded the problems specified by the participants, the number of *Next Solution* requests, the solutions generated by BCDR, and the evaluation scores. Using the problems recorded, we also evaluated how many of them can be solved with a temporal-only configuration after the survey. BCDR found solutions and reached an agreement with the participants in 52 out of 54 sessions. In the solutions for five out of six scenarios, domain relaxation was used for resolving conflicts in the problems specified by the participants (Table 8.3). Compared to the temporal-relaxation only approach, which gave up on 11 sessions, the introduction of domain relaxation indeed provides the users more flexibility and higher chance of finding solutions for their over-subscribed trips.

In general, participants of the user survey found Uhura to be useful in helping them plan daily tasks. They found it simplified the used-to-be complicated and time-consuming planning tasks. Usually, planning a day trip with a few tasks may take

#	Quality	Novelty	Reject& NextSol	Temporal Relaxation	Domain Relaxation
1	3.3 (1.4)	3.8 (1.08)	2.9 (2.3)	2.0 (2.6)	2.1 (2.7)
2	2.4 (1.5)	2.9 (1.22)	3.5 (2.8)	1.3 (2.9)	3.0 (3.3)
3	2.7 (1.5)	3.9 (0.83)	4.7 (5.5)	2.9 (3.0)	3.1 (2.8)
4	3.7 (1.6)	3.8 (1.17)	3.1 (3.0)	0.3 (0.7)	1.7 (3.4)
5	3.2 (1.4)	3.4 (1.20)	2.9 (2.1)	1.9 (2.6)	1.7 (3.0)
6	3.3 (1.5)	3.8 (1.08)	1.2 (0.6)	0.6 (1.1)	0.0 (0.0)

Table 8.3: Average quality and novelty scores, *NextSolution* requests, temporal and domain relaxations (with standard deviation)

minutes or even an hour, depending on the number of alternatives and constraints in the problem. With Uhura, a feasible solution can be found in seconds, and in this survey it often took less than six *Next Solution* iterations before the participants and Uhura agreed on a plan.

In addition to finding feasible resolutions to the conflicts, we are also interested in the quality and novelty of BCDR's solutions. It occasionally produced plans that are new to the participants, as the average novelty scores are above 3 for most scenarios (Figure 8-19). On the other hand, the quality scores indicate that BCDR's solutions are acceptable, but not much preferred, as the average ratings are in the range of 2s and 3s. The scores are lower in scenario 2 and 3, for which more domain relaxations were used in the solutions (average 3.0 and 3.1 per session). This is likely caused by the issues in BCDR's preference model. The results showed that our simple procedure of integrating costs from temporal and domain relaxations does not penalize domain relaxations enough sometimes, which makes BCDR too aggressive in relaxing domains, even in some scenarios where slightly weakening temporal constraints is sufficient. In addition, some participants reported that BCDR lacks of a personalized preference model: it uses the same static cost functions over temporal relaxations, and vector distance models over domain relaxations for all users. if the first plan BCDR returned does not look good, it can hardly find better alternatives afterwards, regardless of how many times they asked Uhura for *Next Solution*. As can be seen in Figure 8-18, the quality scores do not improve as the participants ask for more candidate plans. This is because Uhura enumerates candidate plans in best-first or-

der, and given a poor preference model to start with, it usually cannot come up with better plans that meet the users' needs. While some users find it good at capturing their preferences, others may think BCDR's trade-offs do not make sense at all. This is the cause of the large variance in the quality scores, and is an important problem for future research. One alternative approach is to use a multi-objective preference model, which is likely to perform better in the integration of these different objective functions. Note that it will require a different configuration of BCDR's search queue for enumerating candidates along the pareto-front.

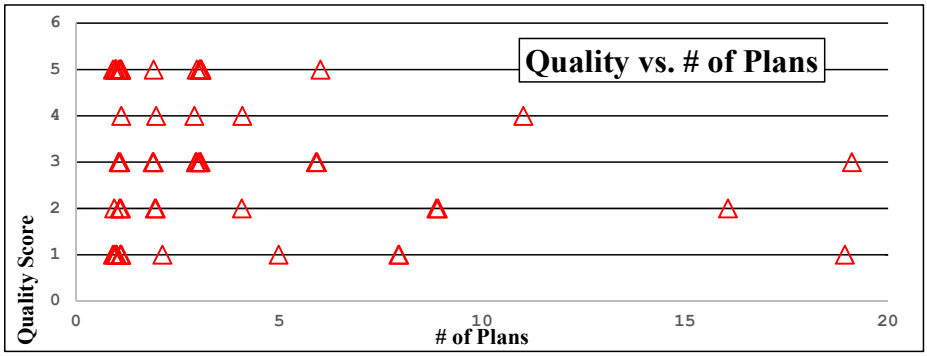


Figure 8-18: Quality score vs. plan requests (X-axis values perturbed to show overlapping data points)

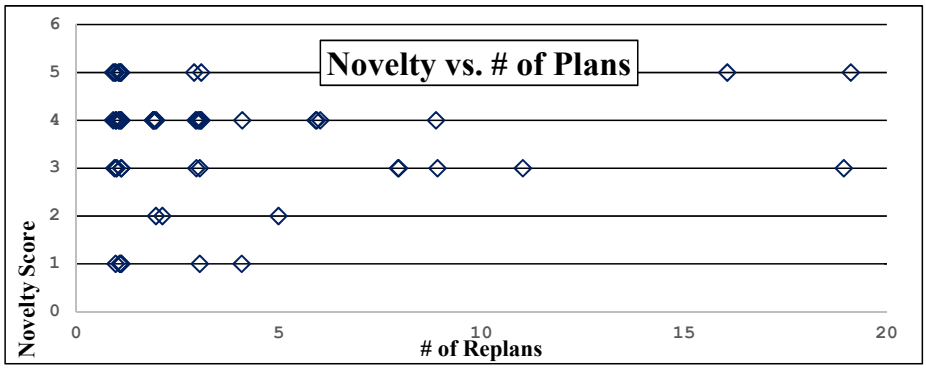


Figure 8-19: Novelty score vs. plan requests (X-axis values perturbed to show overlapping data points)

Finally, as presented in previous sections, the cost of computing domain relaxations is significantly higher than that of temporal relaxation. On average, each knowledge base query takes around 500 ms, and each semantic similarity calculation takes about 200 ms. Due to the size of the knowledge base and word2vec model for

Freebase entities, they were deployed on separate servers from the one for BCDR. The delay in network connection is a big factor that affects BCDR's performance. Therefore, when using BCDR for domain specific applications, one may reduce the coverage of the knowledge base and similarity model for better run-time performance.

8.5 Chapter Summary

In this chapter, we presented the experiment results in the domains of deep-sea mission planning, transit vehicle scheduling, and resource-constrained project scheduling to evaluate BCDR's effectiveness in computing continuous temporal relaxations for resolving large and complex plans. In addition, when evaluated empirically on a range of urban trip planning scenarios, the domain relaxation capability has also demonstrated a substantial improvement in flexibility compared to temporal relaxation only approaches.

Chapter 9

Concluding Remarks

9.1 Summary of Contributions

In this thesis, we presented a conflict-directed relaxation algorithm, called Best-first Conflict-Directed Relaxation (BCDR), for computing continuous relaxations for over-subscribed temporal plans. BCDR reasons with the Temporal Plan Network model to detect the cause of over-subscription, and enumerates relaxations for temporal bounds, chance constraints and variable domains to restore feasibility in best-first order. The key of the continuous relaxation capability is to generalize the discrete conflicts and relaxations used in prior work, to hybrid conflicts and relaxations, which denote minimal inconsistencies and minimal relaxations to both discrete and continuous relaxable constraints.

In this thesis, we focus on solving the following four sub-problems posed by the development of BCDR:

1. The problem of detecting the exact cause of failure in over-subscribed temporal plans, and enumerate their relaxations in best-first order.

2. The problem of computing preferred continuous relaxations, instead of suspensions, for temporal bounds involved in a conflict, based on a user preference model.

3. The problem of generating a robust and risk-bounded relaxations for plans under temporal uncertainty.

4. The problem of computing preferred relaxations, instead of complete removal, for domain constraints, based on a user preference model.

In order to solve these problems, we presented four contributions in this thesis.

1. **A Conflict-directed Relaxation Framework for Over-subscribed Temporal Plans.** Our first contribution is a novel framework for detecting and resolving conflicts in temporal plans, which supports the enumeration of a variary of partial relaxations in best-first order, and the incorporation of user feedback.

2. **Continuous Relaxation for Temporal Constraints.** Our second contribution is a method for computing continuous relaxations, instead of suspensions, for conflicting temporal bounds in over-subscribed temporal plans.

3. **Robust Relaxation Under Temporal Uncertainty.** Our third contribution is an approach for learning conflicts from temporal plans that involve uncertain durations and chance constraints, and computing risk-bounded relaxations for them.

4. **Domain Relaxation for Parameterized Variables.** Our final contribution is a method for computing relaxations for domain constraints, which enables more options to be added into the variable domains, for resolving conflicts in temporal plans.

Contribution 1 provides the basic generate-and-test framework for conflict detection and resolution, and Contribution 2 through 4 are different relaxation techniques implemented as extensions to the relaxation generator and consistency tester. These extensions to the tester provide support for learning conflicts that involve controllable and uncertain temporal durations, while to the generator they support risk-bounded and continuous relaxations for temporal and domain constraints.

The BCDR algorithm has been incorporated as part of a plan relaxation advisor, called Uhura, to help users repair over-subscribed temporal plans in multiple domains, including deep sea explorations, transit route management and urban travel planning. Results from our experiments have demonstrated its efficiency in resolving large and highly constrained logistic problems, and effectiveness in collaborating with humans to resolve urban travel planning problems with a large number of alternatives.

9.2 Future Work

There are a number of interesting avenues for future research on the topic of plan relaxation. We elaborate on some of them here.

A New Relaxation Approach for Generative Planning Problems

Currently, BCDR is only capable of resolving over-subscribed temporal plans, and handling risk-bounds defined over temporal constraint subject to uncertain durations. While useful in some situations, it is not able to help in many real-world scenarios when a plan is not given. BCDR cannot solve the large class of generative planning problems, which are commonly encountered in manufacturing applications. In order to enable such capability, BCDR's conflict learning algorithm must be extended to support a richer activity model with preconditions & effects, and efficiently extract cause of failure from conflicting goals and incomplete planning domains. And the relaxation generator needs to support new types of relaxations, such as introducing additional actions into the planning domain and weakening the preconditions and effects of actions. While BCDR's current conflict learning functions are essentially temporal constraint solvers without any planning capability, the domain relaxation feature presented in Chapter 6 can be viewed as a first step towards this objective, which may be extended for relaxing goal descriptions and action models without much difficulty.

A More Accurate Preference Model over Discrete and Continuous Choices

BCDR takes a hybrid approach to compute both the decisions and relaxation over discrete variables (e.g. alternative plan selection), and continuous relaxations over temporal and chance constraints. It uses a simple weighted preference model over these choices to enumerate candidate solutions in best-first order. While easy to compute and provides support for total ordering, the simple weighted preference often cannot capture the human users true intent. For example, one missing feature that often causes BCDR to be overly conservative is the support for conditional relations between choices. In some situations, the users may prefer to wait for hours at a very nice restaurant, but choose not to visit an ok one even if it has no queue. To effectively computes plan relaxations, BCDR needs a more accurate model to capture human preferences over both discrete and continuous choices, which also supports best-first enumeration. One possible method is to extend the CP-nets (Boutilier, Brafman, Domshlak, Hoos, & Poole, 2004) model for continuous variables, on which (Mohammed, Mouhoub, & Alanazi, 2015) has demonstrated one approach with promising results for online shopping recommendation.

Adopting Partial Constraint Satisfaction Techniques for Improving Efficiency

Finally, improving efficiency for large scale problems is a key issue to address for BCDR. BCDR was designed to enumerate candidates in best-first order, prioritizing quality over speed while generating consistent relaxations. However, in many real-world scenarios, especially in the domain of logistics and manufacturing, it is often impractical to find the best solution due to the large search space and number of constraints. In such scenarios, users prefer to have solutions that are 'ok' but can be found in a bounded period of time. Instead of a complete best-first enumeration strategy, a Partial Constraint Satisfaction (PCS) approach may be more suitable for BCDR in these domains. In the constraint programming literature, many search techniques have been developed for solving large scale logistic problems, including

enforced hill-climbing, large neighborhood search, and column generation. Incorporating them into BCDR will greatly improve its efficiency on large-scale problems. On the other hand, taking an incomplete approach puts a new challenge on plan relaxation, since BCDR may ask the users to relax some of the constraints that may not be necessary. It requires the preference model to handle this type of decision uncertainty, and is an important component of future work.

Appendix A

Definition of Chance Constraints

In the definition below, we use the standard definitions of random variables from probability theory (Durrett, 2010). Intuitively, when describing outcomes of random variables, we can imagine sampling ω from a sample space Ω. The outcomes of random variables can be generated via functions $\mathbf{f}(\omega)$. Further, we have a probability measure P, which takes a subset $A \subseteq \Omega$ and gives the probability of samples being in A.

A chance-constraint is rigorously defined as follows:

Definition 23. *Consider a constraint program with decision variables* $\mathbf{x}$*, and a set of random variables* $\mathbf{f}(\omega)$ *with probability measure* P*, sample space* Ω*, and* $\omega \in \Omega$*.*

Let the set of constraints be defined over decision variables and random variables as

$$C = \bigvee_{i \in I} \bigwedge_{j \in J_i} A_j(\mathbf{x}, \mathbf{f}(\omega)) \geq b_j$$

where I *can be thought of as a set of distinct scenarios, and* J_i *can be thought of as a listing of the set of constraints required to hold in each scenario.*

A chance-constraint *is a tuple* $\langle C, \Delta \rangle$*, requiring:*

$$P(C \text{ satisfied}) \equiv P\left(\left\{\omega \,\middle|\, \bigvee_{i \in I} \bigwedge_{j \in J_i} A_j(\mathbf{x}, \mathbf{f}(\omega)) \geq b_j \right\}\right) \geq \Delta \tag{A.1}$$

While the definition here is presented for constraints systems in disjunctive normal

form, this is a general representation with conversions via De Morgan's laws, and is sufficient for the chance-constrained problem definitions presented in Chapter 3.

Appendix B

Proofs for BCDR's Completeness

In this chapter, we present the proofs for the completeness of BCDR on computing continuous relaxations, and its extension for controllable relaxations (BCDR-U).

B.1 Completeness of BCDR for Computing Continuous Relaxations

Theorem 1. *(Completeness of BCDR) If an over-subscribed TPN can be resolved, then BCDR will return a consistent relaxation that weakens the temporal bounds of some of its episodes.*

Proof. (Proof by contradiction) Given a continuously relaxable TPN $T : \langle P, Q, V, E, RE, L_e, L_p, f_p, f_e \rangle$, assuming that $Sol : \langle A, S, R_e, E' \rangle$ is a valid solution for T, but no consistent relaxation is returned by BCDR.

As presented in Algorithm 5, BCDR only terminates and return *null* if none of the candidate solutions found can resolve all known conflicts C. In other words, if BCDR fails to find a solution, Function ExpandOnConflict must have failed to find a set of relaxations for all conflicts in C. However, such a set of relaxations do exist: Sol is a valid solution for T, meaning that it resolves all conflicts C' in T.

There are only two possible causes of such a situation: (1) the consistency checking algorithm returns conflicts that are not in C', meaning that the conflicts are not valid

and their constraints in fact consistent; and (2) the linear program algorithm we use to compute continuous relaxations is incomplete, in that it failed to find a solution for a feasible LP problem.

For (1), the Bellman-Ford algorithm is proved to be sound and complete. Hence given a network T', it will neither signal failure if T' is consistent, nor return a set of constraints that do not form a negative cycle as a conflict if T' is inconsistent.

For (2), the LP solver we have been using with BCDR are LPSolve (Berkelaar, Eikland, & Notebaert, 2008) and Gurobi (Gurobi Optimization, 2015). Similarly, both solvers are sound and complete on LP problems given sufficient computation time. They will never signal failure on a feasible LP problem, or return a solution that does not respect all the constraints.

Therefore BCDR will not signal failure on an over-subscribed TPN that has a feasible solution. The assumed situation will never occur. □

B.2 Completeness of BCDR-U for Computing Controllable Relaxations

Lemma 1. *Given a STNU that is not strongly controllable, Algorithm 8 returns a* ***minimal conflicting*** *set of constraints.*

Proof. First of all, the triangular reduction algorithm BCDR-U(SC) uses to check strong controllability is proven to be sound and complete. Hence it will not signal failure on strongly controllable network.

Second, we show that given an uncontrollable STNU $T : \langle V, V_r, E, E_u \rangle$, Algorithm 8 will return a **valid** conflicting set of constraints C.

If T is not strongly controllable, then the consistency checking function will return a negative cycle after evaluating the reduced graph. The conflict C is generated by collecting all supporting constraints for the edges in the cycle. All supporting constraints for an edge have been preserved by the generation procedure of the distance graph and the triangular reduction procedure. Even if we are only given the con-

straints in C, we can reconstruct a distance graph that contains the negative cycle. Therefore, constraints in C are guaranteed to be in conflict.

Finally, we show that the conflict, C, returned by Algorithm 8 is **minimal**, meaning that any proper subset of C is not a valid conflict.

Given a negative cycle, only those constraints that contribute to the edge weights in the cycle will be collected into C. If we suspend any constraint in C, the negative cycle will be eliminated. For example, if a requirement edge $c_i \in C$ is suspended, which is equivalent to setting the lower and upper bounds of c_i to $[-\infty, +\infty]$, at least one of the edges in the negative cycle will have weight $+\infty$. If a contingent edge $u_i \in C$ is suspended, then all edges supported by it will have weight $+\infty$, since the reductions that involve u_i not longer exist. Therefore, C is a minimal conflict. □

Lemma 2. *Given a STNU that is not dynamically controllable, Algorithm 9 returns a* ***minimal conflicting*** *set of constraints.*

Proof. Similar to the proof for Lemma , FASTDCCHECK is proven to be sound and complete. Hence it will not signal failure on dynamically controllable network.

For networks that are not dynamically controllable, our conflict extraction procedure guarantees to return a minimal conflicting set of constraints (Algorithm 9): given a negative cycle detected by the ALLMAXCONSISTENCY procedure on the reduced graph, only those constraints that contribute to the edge weights in the cycle will be collected and returned. We show that given an uncontrollable STNU $T : \langle V, V_r, E, E_u \rangle$, Algorithm 9 will return a **valid** conflicting set of constraints C.

If T is not dynamically controllable, then the ALLMAXCONSISTENCY function will return a negative cycle after evaluating the reduced graph. The conflict C is generated by collecting all supporting constraints for the edges in the cycle, plus all constraints contributed to the reductions that produced these edges. Therefore, they form a *semi-reducible negative cycle* in the original graph. Even if we are only given the constraints in C, we can reconstruct a distance graph that produces the negative cycle. Therefore, constraints in C are guaranteed to be in conflict.

Finally, we show that the conflict, C, returned by Algorithm 9 is **minimal**, mean-

ing that any proper subset of C is not a valid conflict.

Only constraints that contribute to the edge weights of the reductions to produce them are collected into C. Therefore, constraints in C forms a minimal semi-reducible negative cycle. If we suspend any constraint in C, the semi-reducible negative cycle will be eliminated, either because some edges created from the suspended constraint will disappear (hence the cycle is no longer negative), or because some reductions will be disabled (hence the negative cycle is no longer semi-reducible). Therefore, C is a minimal conflict. □

Theorem 2. *(Completeness of BCDR-U) If an over-subscribed TPNU can be resolved, then BCDR-U will return a controllable relaxation that weakens the temporal bounds of some of its episodes.*

Proof. (Proof by contradiction) Given a continuously relaxable TPNU $T : \langle P, Q, V, V_r, E, E_u, RE, RE_u, L_e, L_p, f_p, f_e \rangle$, assuming that $Sol : \langle A, S, R_e, R_u, E' \rangle$ is a valid solution for T, but no consistent relaxation is returned by BCDR.

We take a similar approach to prove BCDR-U's completeness by constructing a contradiction. If BCDR-U fails to find a solution, Function ExpandOnConflict must have failed to find a set of relaxations for all conflicts in C. However, *Sol* demonstrates that such a set of relaxations do exist.

There are only two possible causes of such a situation: (1) the controllability checking algorithm returns invalid conflicts that are not in C'; and (2) the linear program algorithm we use to compute continuous relaxations is incomplete. We have proved that (2) never occurs in the proof for Theorem 1. Here we focus on (1) and examine this cause for both BCDR-U(SC) and BCDR-U(DC), which uses different controllability checking algorithms for extracting conflicts.

For BCDR-U(SC), the triangular reduction algorithm it uses to check strong controllability is proven to be sound and complete. Hence it will not signal failure on strongly controllable network. For uncontrollable networks, our conflict extraction procedure guarantees to return a minimal conflicting set of constraints (Lemma B.2).

For BCDR-U(DC), the fastDCcheck algorithm it uses to check dynamic con-

trollability is also proven to be sound and complete. Hence it will not signal failure on dynamically controllable network. Similar to strong controllability, for networks that are not dynamically controllable, our conflict extraction procedure guarantees to return a minimal conflicting set of constraints (Lemma 2).

Therefore BCDR-U will not signal failure on an over-subscribed TPNU that has a feasible solution. The assumed situation will never occur. □

Appendix C

Strong and Dynamic Controllability Checking for STNUs

In this section, we review the algorithms for checking the strong and dynamic controllability of STNUs.

C.1 Checking Strong Controllability using Triangular Reduction

The polynomial time algorithm presented in (Vidal & Fargier, 1999) evaluates if an STNU is *Strongly Controllable* using triangular reductions (Algorithm 18). The reduction procedure adds additional constraints to enforce the satisfaction for the complete range of uncertain durations. Once all reductions complete, the algorithm then checks the consistency of the graph with these additional edges from reduction. If the extended graph is consistent, then the STNU is controllable, meaning that a schedule can be found that is consistent regardless of the outcomes of the uncertain durations.

There are three major steps in this algorithm:

- Map the input STNU to its equivalent distance graph.

Input: A STPU $T = \langle V, E, E_u \rangle$.
Output: A boolean value in $True, False$ that indicates if T is strongly controllable.

```
1  DG ←GetDistanceGraph(T);
2  ReductionQ ←NonContingentEdges(DG);
3  while ReductionQ ≠ ∅ do
4  |  α ←Dequeue(ReductionQ);
5  |  if End(α) is uncontrollable then
6  |  |  β ← ContingentEdgeEndAt(End(α)) //retrieve the contingent
   |  |  edge that ends at the end node of edge α;
7  |  |  α' ← Reduce(α, β);
8  |  |  γ ←GetEdge(Start(α),Start(β));
9  |  |  if Weight(α') <Weight(γ) then
10 |  |  |  γ ← α' //only record the reduction if it results in an edge that is
   |  |  |  tighter than existing ones α;
11 |  |  |  ReductionQ ← ReductionQ ∪ γ;
12 |  |  endif
13 |  else if Start(α) is uncontrollable then
14 |  |  β ← ContingentEdgeStartAt(Start(α)) //retrieve the contingent
   |  |  edge that ends at the start node of edge α;
15 |  |  α' ← Reduce(α, β);
16 |  |  γ ←GetEdge(End(β),End(α));
17 |  |  if Weight(α') <Weight(γ) then
18 |  |  |  γ ← α';
19 |  |  endif
20 |  endif
21 end
22 NCycle ←Bellman-Ford(DG);
23 return NCycle == null;
```

Algorithm 18: Strong controllability checking algorithm

- Reduce all non-contingent edges that end (Line 13) or start (Line 5) at an uncontrollable node using the triangular reduction rule.

- After the reductions, we run the Bellman-Ford algorithm on the reduced graph (Line 24). If no negative cycle is detected, the algorithm returns $True$ to indicate that the input STNU is strongly controllable. Otherwise $False$ is returned.

There are two reduction rules for distance edges from requirement links in the triangular reduction procedure, which different in whether the edge starts or ends at received event. We demonstrate them using the example shown in Figures C-1 and

C-2.

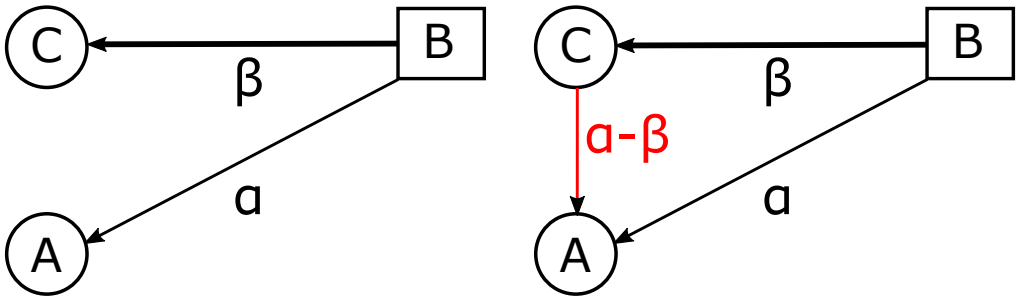


Figure C-1: Reduction rule for edges starting from receive events

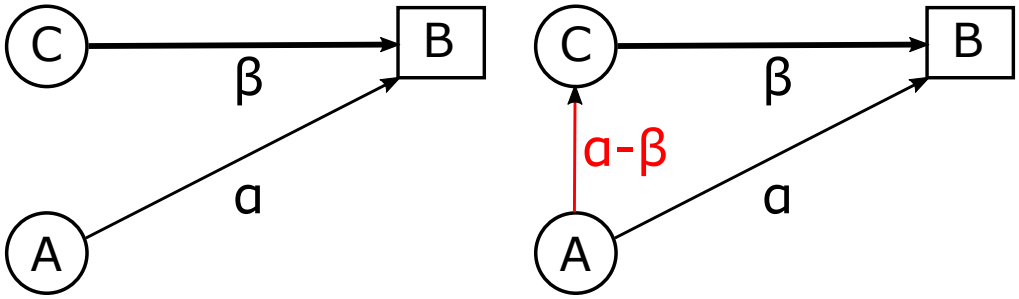


Figure C-2: Reduction rule for edges ending at receive events

- For requirement edge α that starts at a receive event B, we add an additional edge from C to A (Figure C-1). The weight of the new edge is computed by subtracting the weight of the contingent edge β from the weight of α.

- For requirement edge α that ends at a receive event B, we add an additional edge from A to C (Figure C-2). The weight of the new edge is computed by subtracting the weight of the contingent edge β from the weight of α.

During the reduction, if we are trying to add a edge to a pair of events between which a edge already exists, then the reduced edge is only preserved if its weight is smaller than the existing one (Line 9 and 17). In this situation the existing edge can be removed to save memory space since it is dominated by the new edge.

C.2 Checking Dynamic Controllability using FASTDCCHECK

The FASTDCCHECK algorithm presented in (Morris, 2006) evaluates if an STNU is *Dynamically Controllable*. It has a time complexity of $O(N^4)$ and was the fastest DC checking algorithm when introduced. This paper proves that a STNU is dynamically controllable if and only if it does not have a negative cycle that is semi-reducible. A *semi-reducible* path in an STNU is one sequence of distance edges that can be transformed into one without *lower-case* edge. The FASTDCCHECK algorithm utilizes this theorem and determines the dynamic controllability of a STNU by checking if the network contains any *semi-reducible* negative cycle. It iteratively applies a collection of reduction rules to the network and checks its consistency in order to expose any such cycles. The pseudo code of the algorithm is presented in Algorithm 19.

Input: A STPU $T = \langle V, E, E_u \rangle$.
Output: A boolean value in $True, False$ that indicates if T is dynamically controllable.

```
1  DG ←GetNormalDistanceGraph(T);
2  for 1 to K do
3  |  NCycle ←AllMaxConsistent(DG);
4  |  if NCycle == null then
5  |  |  for E in LowerCaseEdges(DG) do
6  |  |  |  moatPaths ←Propagate(E);
7  |  |  |  for Path in moatPaths do
8  |  |  |  |  E' ←Reduce(E,Path);
9  |  |  |  |  AddToGraph(E', DG)
10 |  |  |  end
11 |  |  end
12 |  else
13 |  |  return False;
14 |  endif
15 end
16 NCycle ←AllMaxConsistent(DG);
17 return NCycle == null;
```

Algorithm 19: FASTDCCHECK algorithm for checking dynamic controllability

The key procedure in FASTDCCHECK is to iterate through each lower-case edge in the network and propagate over all allowed paths to search for their *moat edges*.

For a lower-case edge α, an edge β is a moat edge if (1) it has negative weight; (2) the sum of weights for path P:(End(α),End(β)) is negative; and (3) no other edge γ exists in P between α and β such that P':(End(α),End(γ)) is negative.

Once a set of moat edges are identified detected, fastDCcheck then iterate through the path from the lower-case edge to each of the moat edges, called *moat paths*, and tried to reduce the path to one single edge using the five reduction rules (Figure C-3).

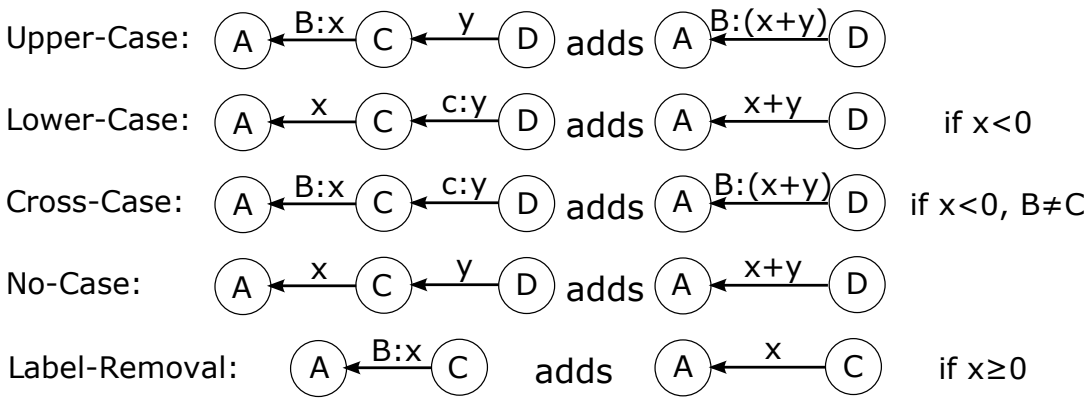


Figure C-3: fastDCcheck reductions

If all reductions are successful, the path will be reduced to an edge and added to the distance graph (Line 8). Once all discovered moat paths for all lower-case edges have been reduced, fastDCcheck runs an *AllMax* consistency check on the graph. The only difference between *AllMax* and a normal consistency check is that the lower-case edges are excluded from the check. If the check fails, fastDCcheck terminates immediately and return false to signal that the input STNU is not dynamically controllable (Line 13) . Otherwise, the algorithm moves on to the next iteration. In total, there could be at most K iterations, where K equals the number of lower-case edges in the distance graph. If none of the AllMax checks fails after K iterations, fastDCcheck will run it for one more time (Line 16) and return true if it succeeds.

Bibliography

Allen, J., & Ferguson, G. (2002). Human-machine collaborative planning. In *Proceedings of the Third International NASA Workshop on Planning and Scheduling for Space*, pp. 27–29.

Armando, A., Castellini, C., & Giunchiglia, E. (1999). Sat-based procedures for temporal reasoning. In *Proceedings of the 5th European Conference on Planning (ECP-1999)*, pp. 97–108.

Bailey, J., & Stuckey, P. (2005). Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In Hermenegildo, M., & Cabeza, D. (Eds.), *Practical Aspects of Declarative Languages*, Vol. 3350 of *Lecture Notes in Computer Science*, pp. 174–186. Springer Berlin / Heidelberg.

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

Bartholdi, J. J., & Eisenstein, D. D. (2012). A self-coördinating bus route to resist bus bunching. *Transportation Research Part B: Methodological*, *46*(4), 481–491.

Beaumont, M., Sattar, A., Maher, M., & Thornton, J. (2001). Solving overconstrained temporal reasoning problems. In *Proceedings of the 14th Australian Joint Conference on Artificial Intelligence (AI-2001)*, pp. 37–49.

Beaumont, M., Thornton, J., Sattar, A., & Maher, M. (2004). Solving overconstrained temporal reasoning problems using local search. In *Proceedings of the 8th Pacific Rim Conference on Artificial Intelligence (PRICAI-2004)*, pp. 134–143.

Bellei, G., & Gkoumas, K. (2010). Transit vehicles headway distribution and service irregularity. *Public transport*, *2*(4), 269–289.

Bellman, R. (1956). On a routing problem. Tech. rep., DTIC Document.

Berkelaar, M., Eikland, K., & Notebaert, P. (2008). *lpsolve : Open source (Mixed-Integer) Linear Programming system.*

Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. H., & Poole, D. (2004). Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, *21*, 135–191.

Cui, J., Yu, P., Fang, C., Haslum, P., & Williams, B. C. (2015). Optimising bounds in simple temporal networks with uncertainty under dynamic controllability

constraints.. In *Proceedings of the Twenty-fifth International Conference on Automated Planning and Scheduling (ICAPS-2015)*, pp. 52–60.

de Kleer, J., & Williams, B. C. (1987). Diagnosing multiple faults. *Artificial Intelligence*, *32*, 97–130.

Dechter, R., Meiri, I., & Pearl, J. (1991). Temporal constraint networks. *Artificial Intelligence*, *49*, 61–95.

Domshlak, C., & Mirkis, V. (2015). Deterministic oversubscription planning as heuristic search: Abstractions and reformulations. *Journal of Artificial Intelligence Research*, *52*, 97–169.

Durrett, R. (2010). *Probability: Theory and Examples*. Cambridge university press.

Effinger, R., & Williams, B. (2005). Conflict-directed search through disjunctive temporal plan networks. *CSAIL Research Abstracts - 2005*, *1*.

Fang, C., Yu, P., & Williams, B. (2014). Chance-constrained probabilistic simple temporal problems. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI-14)*, pp. 2264–2270.

Ford, L. R. (1956). *Network Flow Theory*. Rand Corporation.

Francis, K. G., & Stuckey, P. J. (2014). Explaining circuit propagation. *Constraints*, *19*(1), 1–29.

Freed, M., Carbonell, J., Gordon, G., Hayes, J., Myers, B., Siewiorek, D., Smith, S., Steinfeld, A., & Tomasic, A. (2008). Radar: A personal assistant that learns to reduce email overload. In *AAAI*.

Freuder, E. C., & Wallace, R. J. (1992). Partial constraint satisfaction. *Artificial Intelligence*, *58(1-3)*, 21–70.

Graphhopper (2015). Graphhopper directions api with route optimization. `https://graphhopper.com/`. Accessed: 2015-11-15.

Gunopulos, D., Khardon, R., Mannila, H., & Sharma, R. S. (2003). Discovering all most specific sentences. *ACM Transactions on Database Systems*, *28*, 140–174.

Gurobi Optimization, I. (2015). Gurobi optimizer reference manual..

Haklay, M., & Weber, P. (2008). Openstreetmap: User-generated street maps. *Pervasive Computing, IEEE*, *7*(4), 12–18.

Hooker, J. N., & Ottosson, G. (2003). Logic-based benders decomposition. *Mathematical Programming*, *96*(1), 33–60.

Kall, P. (1976). Chance constrained programming. In *Stochastic Linear Programming*, pp. 79–92. Springer.

Kemp, C., & Tenenbaum, J. B. (2008). The discovery of structural form. *Proceedings of the National Academy of Sciences*, *105*(31), 10687–10692.

Khatib, L., Morris, R., Morris, R., & Rossi, F. (2001). Temporal constraint reasoning with preferences. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, pp. 322–327.

Kilby, P., & Shaw, P. (2006). Vehicle routing. In Rossi, F., Beek, P. V., & Walsh, T. (Eds.), *Handbook of Constraint Programming*, pp. 801–836. Elsevier B.V.

Kim, P., Williams, B. C., & Abramson, M. (2001). Executing reactive, model-based programs through graph-based temporal planning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-01)*, pp. 487–493.

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

Li, H., & Williams, B. (2005). Generalized conflict learning for hybrid discrete/linear optimization. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming*, pp. 415–429.

Liffiton, M., Moffitt, M., Pollack, M., & Sakallah, K. (2005). Identifying conflicts in overconstrained temporal problems. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, pp. 205–211.

Liu, J., Cyphers, S., Pasupat, P., McGraw, I., & Glass, J. (2012). A conversational movie search system based on conditional random fields.. In *INTERSPEECH*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013a). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119.

Mikolov, T., Yih, W.-t., & Zweig, G. (2013b). Linguistic regularities in continuous space word representations.. In *HLT-NAACL*, Vol. 13, pp. 746–751.

Moffitt, M. D., & Pollack, M. E. (2005a). Applying local search to disjunctive temporal problems. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-2005)*, pp. 242–247.

Moffitt, M. D., & Pollack, M. E. (2005b). Partial constraint satisfaction of disjunctive temporal problems. In *Proceedings of the 18th International Florida Artificial Intelligence Research Society Conference (FLAIRS-2005)*, pp. 715–720.

Mohammed, B., Mouhoub, M., & Alanazi, E. (2015). Combining constrained cp-nets and quantitative preferences for online shopping. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pp. 702–711. Springer.

Morris, P. (2006). A structural characterization of temporal dynamic controllability. In *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP-2006)*, pp. 375–389.

Morris, P. (2014). Dynamic controllability and dispatchability relationships. In *Integration of AI and OR Techniques in Constraint Programming*, pp. 464–479. Springer.

Morris, P., & Muscettola, N. (2005). Temporal dynamic controllability revisited. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-2005)*, pp. 1193–1198.

Noessner, J., Martin, D., Yeh, P., & Patel-Schneider, P. (2015). Cogmap: A cognitive support approach to property and instance alignment. In *ISWC*.

Ortiz, C., & Shen, J. (2014). Dynamic intention structures for dialogue processing. In *Proceedings of the 18th Workshop on the Semantics and Pragmatics of Dialogue (SemDial 2014)*.

O'Sullivan, B., Papadopoulos, A., Faltings, B., & Pu, P. (2007). Representative explanations for over-constrained problems. In *AAAI 07*, pp. 323–328.

Peintner, B., Moffitt, M. D., & Pollack, M. E. (2005). Solving over-constrained disjunctive temporal problems with preferences. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS 2005)*.

Peintner, B., Venable, K. B., & Yorke-Smith, N. (2007). Strong controllability of disjunctive temporal problems with uncertainty. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP-2007)*, pp. 856–863.

Raiman, J. (2016). Building blocks for the mind. Master's thesis, Massachusetts Institute of Technology.

Rossi, F., Sperduti, A., Venable, K. B., Khatib, L., Morris, P. H., & Morris, R. A. (2002). Learning and solving soft temporal constraints: An experimental study. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, pp. 249–263.

Rossi, F., Venable, K. B., & Yorke-Smith, N. (2006). Uncertainty in soft temporal constraint problems: A general framework and controllability algorithms forthe fuzzy case. *Journal of Artificial Intelligence Research*, *27*, 617–674.

Stauber, B. R., Douty, H., Fazar, W., Jordan, R. H., Weinfeld, W., & Manvel, A. D. (1959). Federal statistical activities..

Stergiou, K., & Koubarakis, M. (1998). Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, *120*, 248–253.

Thompson, C. A., Goker, M. H., & Langley, P. (2004). A personalized system for conversational recommendations. *Journal of Artificial Intelligence Research*, *21*, 393–428.

Tsamardinos, I. (2002). A probabilistic approach to robust execution of temporal plans with uncertainty. In *Methods and Applications of Artificial Intelligence: Proceedings of the Second Hellenic Conference on Artificial Intelligence*, Vol. 2308 of *Lecture Notes in Computer Science*, pp. 97–108.

Venable, K. B., & Yorke-Smith, N. (2005). Disjunctive temporal planning with uncertainty. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, pp. 1721–1722.

Vidal, T., & Fargier, H. (1999). Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence*, *11*, 23–45.

Wah, B. W., & Xin, D. (2007). Optimization of bounds in temporal flexible planning with dynamic controllability. *International Journal on Artificial Intelligence Tools*, *16*(1), 17–44.

Wächter, A., & Biegler, L. T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, *106*, 25–57.

Williams, B. C., & Ragno, R. J. (2002). Conflict-directed A* and its role in model-based embedded systems. *Journal of Discrete Applied Mathematics*, *155(12)*, 1562–1595.

Word2Vec (2013). Tool for computing continuous distributed representations of words. `https://code.google.com/archive/p/word2vec/`. Accessed: 2016-01-29.

Yeh, P., Ramachandran, D., Douglas, B., Ratnaparkhi, A., Jarrold, W., Provine, R., Patel-Schneider, P., Laverty, S., Tikku, N., Brown, S., Mendel, J., & Emfield, A. (2015). An end-to-end conversational second screen application for tv program discovery. *AI Magazine*, *36*(3).

Yorke-Smith, N., Saadati, S., Myers, K., & Morley, D. (2012). The design of a proactive personal agent for task management. *IJAIT*, *21*(1).

Yu, P., Fang, C., & Williams, B. (2014). Resolving uncontrollable conditional temporal problems using continuous relaxations. In *Proceedings of the Twenty-fourth International Conference on Automated Planning and Scheduling (ICAPS-2014)*, pp. 341–349.

Yu, P., Fang, C., & Williams, B. (2015). Resolving over-constrained probabilistic temporal problems. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI-15)*.

Yu, P., Shen, J., Yeh, P. Z., & Williams, B. (2016a). Resolving over-constrained conditional temporal problems using semantically similar alternatives. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI-16)*, pp. 3300–3307.

Yu, P., Shen, J., Yeh, P. Z., & Williams, B. (2016b). Towards personal assistants that can help users plan. In *Proceedings of the 16th International Conference on Intelligent Virtual Agents (IVA 2016)*.

Yu, P., & Williams, B. (2013). Continuously relaxing over-constrained conditional temporal problems through generalized conflict learning and resolution. In *Proceedings of the 23th International Joint Conference on Artificial Intelligence (IJCAI-2013)*, pp. 2429–2436.