Hierarchical Planning and Scheduling of Looping Activities for Robotic Scouts

by

James G. Paterson

B.Sc. Mechatronics, University of Cape Town (2011)

Submitted to the Department of Aeronautics and Astronautics in partial fulfillment of the requirements for the degree of Master of Science in Aeronautics and Astronautics at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY

[](September 2014)

August 2014

© Massachusetts Institute of Technology 2014. All rights reserved.

Signature redacted

Author .................. ..............................

Department of Aeronautics and Astronautics August 8, 2014

Signature redacted

Certified by .................. ..............................

Brian C. Williams
Professor of Aeronautics and Astronautics
Thesis Supervisor

Signature redacted

Accepted by .................. ..............................

Prof. Paulo C. Lozano Chair, Graduate Program Committee
Hierarchical Planning and Scheduling of Looping Activities for Robotic Scouts

by

James G. Paterson

Submitted to the Department of Aeronautics and Astronautics on August 8, 2014, in partial fulfillment of the requirements for the degree of Master of Science in Aeronautics and Astronautics

Abstract

A wide range of robotic missions contain activities that exhibit looping behaviour. Examples of these activities include picking fruit in agriculture, pick-and-place tasks in manufacturing, or even search patterns in robotic observing missions. These looping activities often have a range of acceptable loop values and a preference function over them. For example, during robotic survey missions, information gain is expected to increase with the number of loops in a search pattern. Since these looping activities also take time, which is typically bounded, there is a challenge of maximizing utility while respecting time constraints. While current scheduling techniques allow us to specify disjunctive temporal constraints and preference over time, they do not allow us to represent looping constraints, or preference over the number of loops.

In this thesis, we provide a capability to optimally choose between multiple candidate plans by selecting threads of execution and the number of loops within each activity, while respecting temporal constraints.

To achieve this, we first present a new scheduling problem; the looping temporal problem with preference (LTPP), as a formalism for encoding scheduling problems that contain looping activities, and provide an algorithm to solve it. The LTPP expresses temporal and looping constraints in a compact form, while adding a preference function on the number of loops between two temporal events. Second, we enable hierarchical temporal planning over looping activities through an optimal temporal planner that uses the LTPP scheduler at its core. This planner takes a looping temporal plan network (LTPN) as an input and produces a consistent, least-commitment temporal plan that optimizes the global value of all preference functions. This least-commitment plan can then be dispatched to the robot online allowing for robustness to temporal disturbances during execution. We demonstrate the capabilities of these algorithms on problems from the search-and-rescue domain.

Thesis Supervisor: Brian C. Williams
Title: Professor of Aeronautics and Astronautics
Acknowledgments

I would like to express my gratitude to all those who have supported me in completing this thesis.

- Firstly, I would like to thank my advisor, Prof. Brian Williams. In his professional capacity and outside of it, his dedication to his students is something I really admire. I have been lucky to have him as a mentor and I have learnt more from him than I could have hoped for. Without him this work would not be possible.

- I’d like to thank everyone in the Model-based and Embedded Robotic Systems group (MERS). It has been such a great atmosphere to work in for the past two years. A special thank you must go to Eric Timmons, whom I have collaborated closely with during my time here, and who has taught me a lot. I also need to thank David Wang and Peng Yu, for helping me get settled in at MIT, and for answering countless algorithmic and coding questions. Others I need to thank for insightful comments and discussions include: Enrique Fernandez, Andrew Wang, Steve Levine, Erez Karpas, Ameya Shroff, Dan Strawser, Cheng Fang, Pedro Santana, Jonathan Raiman, Szymon Sidor, Spencer Lane and Larry Bush.

    I will always remember RMPL, Lisp, TPNs, STNs, STCs, QSPs, CCTPs, and of course LTPPs!

- Lastly, but very importantly, I would like to thank my family, who have given me their unconditional love and support to pursue my dreams.
## Contents

1 Introduction

1.1 Motivating Example: Autonomous Scouts for Search-and-Rescue 13
1.2 Related Work 14
1.3 Solution Method and Contributions 16
1.4 Experimental Results 19
1.5 Thesis Outline 20

2 Problem Statement

2.1 Specifying Looping Activities 23
2.1.1 Specifying Search Patterns as Looping Activities 25
2.2 Encoding Multiple Candidate Solutions in a Looping Temporal Plan Network 27
2.3 Candidate Plan Generation 30
2.4 Optimal Scheduling to produce a Looping Temporal Plan 32
2.5 Plan Reformulation to Dispatchable Form 35
2.6 Chapter Summary 37

3 Optimal Scheduling of Looping Activities

3.1 Autonomous Scouts for Search-and-Rescue 39
3.2 Defining the Looping Temporal Problem with Preference (LTPP) 41
3.3 Candidate Solution Methods 44
3.3.1 Scheduling Community 44
3.3.2 Mathematical Optimization Community 45
3.4  Our Approach to Solving the LTPP  ........................................ 47
  3.4.1  Domain Filtering .................................................. 47
  3.4.2  Searching for the Optimal Solution .................................. 51
3.5  Chapter Summary .................................................. 56

4  Planning over Looping Activities ...................................... 57
  4.1  Clarifying Example .................................................. 59
  4.2  Modelling the Problem ............................................... 61
  4.3  Previous Work ........................................................ 63
  4.4  Extracting an Optimal Looping Plan from a LTPN ................. 64
  4.5  Forming a Dispatchable Plan ....................................... 69
  4.6  Chapter Summary .................................................. 71

5  Experimental Validation ................................................... 73
  5.1  Scheduling of Looping Activities .................................... 73
      5.1.1  Experiment Setup ............................................... 74
      5.1.2  Hypothesis ...................................................... 75
      5.1.3  Results ......................................................... 75
  5.2  Planning over Looping Activities .................................... 77
      5.2.1  Experiment Setup ............................................... 77
      5.2.2  Hypothesis ...................................................... 78
      5.2.3  Varying the Number of Looping Episodes ...................... 79
      5.2.4  Varying the Number of Decision Variables ................... 80

6  Conclusion and Future Work ............................................. 83
  6.1  Future Work ....................................................... 83
      6.1.1  Increase Speed of Optimal Plan Extraction .................. 83
      6.1.2  Optimal Scheduling of Nested Loops .......................... 85
  6.2  Conclusion ........................................................ 86
List of Figures

1-1 Search-and-rescue mission example. ........................................ 15
1-2 Overall planning architecture. .................................................. 17

2-1 Inputs and outputs to planning sub-problems. ...................... 22
2-2 The looping temporal constraint. ............................................. 25
2-3 The loops in common search patterns. ..................................... 26
2-4 Flexibility in the number of loops in a search pattern. .......... 26
2-5 Modelling search patterns as temporal constraints. ................. 27
2-6 A general looping episode. ...................................................... 28
2-7 LTPN representation of a UAV search-and-rescue mission. ......... 29
2-8 A choice in a LTPN. .............................................................. 31
2-9 A complete set of choices in a LTPN. ..................................... 31
2-10 The optimal scheduling problem. ........................................... 33
2-11 An executable looping activity. ............................................... 33
2-12 A schedule for a looping plan. ............................................. 35
2-13 The plan reformulation problem. ........................................... 36

3-1 Optimal scheduling example: Search for two missing hikers. .... 41
3-2 A simple temporal constraint as a special case of a looping temporal constraint. ....................................................... 43
3-3 The search-and-rescue scheduling problem. .......................... 43
3-4 The TCSPP representation of a looping activity. .................... 45
3-5 The MINLP encoding of a LTPP. ............................................ 46
3-6 The input and output of DOMAIFILTER running on the example problem. 48
3-7 The loop pruning process. 50
3-8 After running DOMAIFILTER, no solution is guaranteed. 51
3-9 Search tree for solving a looping temporal problem with preference. 53
3-10 STP representation of candidates. 55
3-11 Solution to the search-and-rescue scheduling problem. 56
4-1 The planning system architecture. 58
4-2 Search-and-rescue planning example. 60
4-3 A general looping episode 61
4-4 Planning problem framed as a looping temporal plan network. 62
4-5 A LTPN vs. a TPN. 63
4-6 Utility upper bound in a LTPN. 66
4-7 Search tree during LTPN optimization. 67
4-8 Calculating heuristic values during candidate generation. 68
4-9 Solution to the planning problem. 69
4-10 A dispatchable temporal plan. 70
5-1 LTPP structure for benchmarking. 74
5-2 Scheduling run-time results. 76
5-3 Scheduling space requirements. 77
5-4 LTPN problem structure for benchmarking. 78
5-5 Plan optimization run-time. 79
5-6 Percentage of problems solved during plan benchmarking. 80
5-7 Effect of decision variables on run time. 81
5-8 The effect of looping temporal constraints in series and in parallel. 82
6-1 Early scheduling can improve optimal plan search. 84
6-2 An example of a conflict. 85
Chapter 1

Introduction

There is a growing need for autonomous robotic systems that can perform tasks that exhibit looping behavior. Unmanned aerial vehicles (UAVs) use patterns that contain iterations (or loops) to map agricultural land to provide farmers with crop estimates and locate areas of disease [5]. Elsewhere, autonomous underwater vehicles (AUVs) perform patterned search of the sea floor to search for hydrothermal vents and other interesting scientific events [12]. Search-and-rescue missions are another common area where UAVs perform patterned search [16] [4]. Outside of searching and mapping, there is a need for robots that can perform repetitive tasks, such as manufacturing pick-and-place tasks on an assembly line or picking fruit in agriculture [19]. Inherently, there is a preference over the number of loops to perform in a repetitive task; during robotic survey missions, the information gain is expected to increase with the number of loops in a search pattern. In all these tasks timing is crucial, thus scheduling is important. For example, if an AUV is mapping the ocean floor using search patterns [7], it needs to ensure completion of tasks by designated time points to ensure the mission is completed before its battery runs out.

Traditionally, activities are specified to a robot with a fixed number of loops, for example, perform a search pattern with 10 loops (or passes in a lawnmower pattern). However, we really need an autonomous system to optimize the selection of these loops as they can significantly influence the overall utility of the mission, for example information gain in observing missions. Selecting the number of loops in each activity
influences the overall duration of that activity (and thus the overall mission duration), which in turn influences the selection of loops in other activities in the mission. Thus, finding the optimal trade off between loops in each activity, while respecting temporal constraints, is a difficult multidimensional optimization problem that is best solved by algorithms, rather than through manual calculation.

Apart from selecting the number of loops within each activity, the autonomous executive should be able to select which types of activities to perform and in what order to ensure the optimal mission. In a search-and-rescue mission, this might correspond to selecting which areas to search and which search patterns to perform within them. Thus, there is a need for temporal planning over looping activities.

Finally, the output of a plan executive for mobile robots should be a temporally-flexible plan. A temporally-flexible plan is one where the time at which each activity starts may be chosen from a range of consistent times. The result is that online dispatching executives [8] can delay the decision of when to start an activity until it is actually time to perform that activity. This allows an online execution system that is dispatching a temporally-flexible plan, to have the ability to handle some temporal disturbances that may occur in a real-world the mission.

The central role of this thesis is to extend optimal, temporally-flexible planning [11] to reason over activities that contain loops and preference over the number of loops (henceforth called looping activities). By reasoning we mean: 1) selecting loop parameters and durations, given a plan with looping activities; 2) checking consistency of a candidate plan with loops, based on whether a feasible set of loop parameters and durations exist, and finally 3) guiding a hierarchical planner to generate an optimal plan with loops. To achieve this, we first show how temporal constraint networks and scheduling algorithms can be extended to looping activities through a simple generalization of simple temporal problems (STPs) [10] to looping temporal problems with preference (LTPPs), where the preference is a function of the number of loops. Next, we present an optimal scheduler for the LTPP that determines the number of loops to perform within each looping activity and consistent bounds on their durations. The scheduler leverages the structure of the problem to improve run-time and space
requirements, allowing it to be used in online, embedded systems. Next, we present an optimal hierarchical temporal planner that uses the LTPP scheduler at its core. The output is a temporal plan with loops (henceforth a looping plan), and although we fix the number of loops, we leave bounds on the loop durations. This allows us to convert the output to a form that can be dispatched online. Finally, we demonstrate the capabilities of these algorithms on problems from a search-and-rescue domain.

The remainder of this chapter is organized as follows. First, we motivate the need for temporal planning over looping activities through an example from the search-and-rescue domain. Next, we briefly discuss some related work that is relevant for looping activities, before presenting our approach to solving the problem and key contributions. Finally, we provide an overview of empirical results and discuss the outline for the rest of this thesis.

1.1 Motivating Example: Autonomous Scouts for Search-and-Rescue

Using aerial vehicles to gather information during observing missions such as search-and-rescue [16] or search-and-tracking [4] is an area of active research. The work in this thesis allows scouting systems to reason over activities that contain loops and an associated preference function over the number of loops. In a search-and-rescue or mapping mission, these activities may be search patterns, and this work provides a way to optimally choose between search patterns and choose the number of loops in each pattern, while adhering to temporal constraints. To help explain concepts and solution methods presented in this thesis, we introduce an example from a search-and-rescue domain. The example is given here at a high level, and depth is added as needed in the following chapters.

Consider the following scenario, where two hikers have not reached their destination on time. From the topology of the area and the route they were expected to take, they are most likely to be in Area A, Area B, or Area C (from now on just called A,
B and C), or on the paths between A and B, or A and C (See Figure 1-1). In order to locate the hikers, a UAV is sent out and is equipped to perform search patterns to search each area. The search patterns are parametrized with a range of loops to choose from, and a controllable temporal duration per loop. For each pattern, there is a utility function that maps the number of loops to a utility, where the utility is calculated according to the belief of where the hikers are. There are also temporal constraints on the mission; the UAV has a maximum flight time of 50 minutes and a flight corridor between A and B briefly opens up between 25 and 35 minutes after the mission starts, allowing transition from A to B. Finally, a choice must be made to search A and B, or A and C, but not search all three areas.

Since, in this example we assume the utility is calculated according to the belief of where the hikers are, then maximizing this utility while maintaining temporal consistency corresponds to the best chance of a successful rescue. In the process of maximizing overall mission utility, we need to:

- choose which areas to search i.e. A and C, or A and B.
- choose which type of search patterns to perform.
- select the number of loops to perform within each search pattern.
- maintain temporal consistency and calculate consistent temporal bounds for loop durations and temporal constraints.

A solution to the problem is shown in Figure 1-1, containing a choice between search areas, choices of search pattern type, as well as integer assignments to loop ranges of each search pattern and temporally consistent bounds on the per loop durations and temporal constraints. We discuss the framing of these problems more precisely in Chapters 3 and 4.

1.2 Related Work

Model-based executives elevate the commanding of autonomous systems to the level of goal states while providing guarantees of correctness [27]. This allows a user to
PROBLEM:

START: $t = 0$

Path only traversable between $t = 25$ and $t = 35$

END: $t \leq 50$

SOLUTION:

START: $t = 0$

25 $\leq t \leq 35$

END: $t \leq 50$

Figure 1-1: Search-and-rescue mission example.
specify goal states and have the planning executive compose a program of activities to achieve those goals. Preference may be added [11] in order to achieve the user's goals optimally. Elsewhere, execution algorithms have been explored that perform scheduling, resource assignment and contingency selection, by making decisions on line [21][8]. While these techniques make a significant advance in finding an optimal program to satisfy a user's goals, they have not addressed activities with loops.

Previous work has been pursued that supports looping behavior such as fixed search patterns during autonomous robotic scouting missions, but these looping behaviors are not parametrized to allow the executive to select the number of loops [4]. Scheduling problems such as temporal constraint satisfaction problems with preference (TCSPPs) [10] allow disjunctive temporal constraints with preference functions over time, but they do not allow us to represent looping ranges, or specify preference over the number of loops without specifying a disjunctive bound and a preference value for each loop.

We discuss related work more deeply in Chapters 3 and 4, in the context of our proposed representations and algorithms.

1.3 Solution Method and Contributions

We frame our problem within a looping temporal plan network (or LTPN), an extension of a temporal plan network [18][11]. A LTPN compactly encodes multiple candidate solutions to the problem and contains events representing points in time, episodes representing time-evolved looping activities and temporal constraints restricting durations between events. We encode choices as decision variables, where an assignment to the domain of a decision variable activates a sub-graph in the LTPN, selecting a thread of execution in the plan. Each looping activity contains a controllable, integer range of loops, a controllable temporal bound per loop, as well as a utility function over the number of loops.

Our goal is to find an assignment to the decision variables and loop ranges of each looping activity, so as to maximize the overall utility of the plan, while maintaining
Input: Looping Temporal Plan Network

Add Candidate

1. Candidate Generator

Optimal Plan?

No

Best Candidate

2. Optimal Scheduler

Yes

Optimal looping plan

3. Plan Reformulation

Output: Dispatchable Temporal Plan

Dispatching

Commands

Observations

Figure 1-2: Overall planning architecture.
temporal consistency. We return a temporal plan containing those choices, along with consistent temporal bounds on loop durations and temporal constraints, so as to ensure a consistent schedule exists.

Our approach to solving the problem is to extend optimal temporally-flexible planning methods [18] [11] to include reasoning over looping activities to find an optimal plan from many candidates. The approach contains three main parts and can be visualized in Figure 1-2:

1. **Candidate Generator**: The first step is to generate a complete set of assignment to decision variables (a complete candidate) by searching through the hierarchical graph in an A*-like manner, making assignments to decision variables. We use the upper bound on utility as an admissible heuristic to make these assignments. In the search-and-rescue example, this would correspond to choosing which areas to search, which activities to perform, and in what order.

2. **Optimal Scheduler**: Once a complete candidate is found, it needs to be checked for consistency. If consistent, the optimal number of loops and consistent temporal bounds need to be calculated, as well as the utility of the candidate. This is done by first employing a domain filtering technique to prune inconsistent loops and then using a novel best-first search technique that searches over ranges of loops at a time to calculate the optimal loop assignments and the overall utility of the candidate.

   If the utility of a candidate after scheduling is higher than the heuristic value of any other candidate on the queue, it is the optimal looping plan. If it is not higher than some other candidate, it is sent back to the candidate generator for re-sorting.

3. **Plan Reformulation**: After finding the optimal looping plan, we reformulate the plan through a process of “loop flattening” into a form that can be dispatched by an online dispatcher. Loop flattening expands each looping activity into a series of sub-activities.
The main focus of this thesis is on Parts 1, 2 and 3 above. Detailed explanation of online dispatching and execution monitoring can be found in the relevant literature [8][20], but is outside the scope of this thesis. Since part 2 is a sub-routine to part 1, it is developed first in Chapter 3, before parts 1 and 3 are developed in Chapter 4.

To enable fast and optimal temporal planning over looping activities, this thesis makes the following key contributions, required to speed up the search and reduce the number of candidate plans searched through before finding the optimal:

- We frame optimal plan selection within a looping temporal plan network (LTPN) (Chapter 2).

- We introduce an optimal heuristic search algorithm to find an optimal and consistent solution to the LTPN (Chapter 4).

- We introduce a new problem, the looping temporal problem with preference (LTPP) to encode temporal problems that contain looping activities (Chapter 3). The LTPP forms a sub-problem to finding an optimal looping plan in the LTPN, once choice has been removed from the problem.

- We solve the LTPP by introducing a novel constraint propagation algorithm to reduce the space of possible solutions and a heuristic best-first search algorithm that searches over ranges of loops and checks consistency incrementally to find an optimal solution efficiently (Chapter 3).

1.4 Experimental Results

We report the results of our experimentation into two sections. First, we benchmark our LTPP solver against a best-first enumeration search and a MINLP solver (called SCIP [2]) on randomly generated search-and-rescue problems. We show improvement in run-time of multiple orders of magnitude as the complexity of the problem grows. We also show that given 10 minutes of run-time, our algorithm can solve problem instances 5 orders of magnitude larger than best-first search, and 2 orders of magni-
tude larger than SciP. We also show more than an order of magnitude improvement in space use (size of the search queue) over best-first as the problem scales.

Then, we demonstrate the run-time improvement made by our algorithm over Kirk [11] for optimal, flexible temporal planning over looping activities. As the size of the problem scales, our new approach solves nearly 50% more problems than Kirk within a 2 minute cut-off. We also investigate the effect of the number of decision variables in the plan graph, showing that the problem becomes easier to solve as the number of decision variables increase.

1.5 Thesis Outline

The remainder of this thesis is organized as follows:

Chapter 2 develops the problem statement and defines concepts and structures that are used to frame the inputs and outputs to our problem.

Chapter 3 introduces the looping temporal problem with preference (LTPP) as a formalism for encoding temporal problems that contain activities with iteration and presents our solution method for it. Note that a LTPP contains no choice between constraints. This corresponds to Part 2 in the planning architecture in Figure 1-2.

Chapter 4 describes our method for finding an optimal looping plan within the looping temporal plan network, that uses the LTPP solver as a sub-routine. The process of loop flattening is also described to allow online dispatching. Thus Chapter 4 completes the planning executive visualized in Figure 1-2.

Chapter 5 presents experiments to empirically validate our approaches above.

Chapter 6 concludes the thesis and presents ideas for future work.
Chapter 2

Problem Statement

The capability we aim to provide is the dynamic execution of a looping temporal plan network (LTPN). Since dynamic execution of robotic systems already exists [8][20], we focus on the problem of mapping a LTPN to the optimal dispatchable temporal plan (without loops). Thus, the output of our system is the input to an online execution system (or dispatching executive). The problem of mapping a LTPN to a dispatchable temporal plan is a large one, thus we divide the problem into three sub-problems (Parts 1-3 in Figure 2-1) and describe each problem separately. This leads to three formal statements: the candidate plan selection problem, the preferred schedule problem and the reformulation of a looping plan to a dispatchable temporal plan. These three problems are described below.

Candidate Generation: The input to the candidate generation problem is a looping temporal plan network (LTPN); a compact encoding of many candidates for the optimal looping plan. A LTPN contains conditional decision variables, where an assignment to the domain of a decision variable selects a thread of execution. The goal is to make a complete set of assignments to the decision variables in the LTPN, thus selecting a thread of execution from the start event to the end event. This is called a Candidate Plan.

Optimal Scheduling: Although a thread of execution has been selected in a Candidate Plan, the optimal number of loops for each looping activity has not yet been selected. Thus the optimal scheduling problem is to assign the optimal number of loops to each
looping activity while maintaining temporal consistency of the plan. This output form is called a looping temporal plan, or looping plan.

**Plan Reformulation:** Finally, once the optimal looping plan has been found, it needs to be reformulated into a dispatchable temporal plan that can then be sent to an online dispatching executive. Note that the dispatchable temporal plan is not in minimal dispatchable form [8] as this is done by the dispatching executive.
In summary, given a LTPN, our problem is first to find the optimal looping plan, and then compile it to a dispatchable temporal plan. In doing so, the following steps need to be performed while optimizing the plan and maintaining temporal consistency:

1. Make assignments to decision variables to form a candidate plan.
2. Make integer assignments to loop variables to form a looping plan.
3. Compile the looping plan to a dispatchable temporal plan with temporal bounds tightened to a minimal network.

The structure of this chapter is as follows. First, we describe looping activities and their encoding of search patterns (Section 2.1). Second, we describe the input to the overall problem, the looping temporal plan network (Section 2.2). Next, we describe the inputs, outputs and concepts of the three sub problems: candidate generation, optimal scheduling and plan reformulation (Sections 2.3-2.5 respectively). Finally, we conclude the chapter in Section 2.6.

2.1 Specifying Looping Activities

A looping activity is an activity in which a sub-activity is repeated some number of times. The number of times the sub-activity is repeated is chosen from a controllable, integer-valued range and each sub-activity has a controllable, real-valued temporal bound on its duration. A looping activity also contains a preference over how many times its sub-activity is repeated. In words, a looping activity might be to: “Perform a spiral search pattern in Area B with between 5 and 20 loops, performing each one within 2 to 3 minutes, collecting a utility of 6 per loop.”

We concern ourselves only with the temporal aspects of these activities, for example, as opposed to their conditions and effects as in generative planning.

Definition 1. Looping Activity: A looping activity between two temporal events, \(x_i\) and \(x_j\) is specified by a tuple \(<\text{command}(), \text{LTC}>\) where:
• The command is a method that can be directly interpreted by the robot and is of the form: command(p₁…pₙ), where p₁ to pₙ are parameters passed to the method.

• LTC is a looping temporal constraint (LTC) that describes the duration of the activity as defined in Definition 2.

Definition 2. Looping Temporal Constraint: A looping temporal constraint (LTC) is a simple temporal constraint repeated between two time events, xᵢ and xⱼ, and is specified by a tuple < Nᵢⱼ, δ, f > where:

- δ is the duration of a simple temporal constraint between the start and end times of one loop, and is of the form δ ∈ [δᵢ, δₜ], δ ∈ R+. This is known as the loop duration.

- N is an integer loop variable between the events xᵢ and xⱼ specifying how many loops to execute. The loop variable is constrained in a loop range: N ∈ [Nᵢ, Nₜ], N ∈ Z⁺.

- f is a monotonic preference function that maps the number of loops N to a real utility value, f : N → R⁺.

The duration bound of a looping activity is described by a LTC and is found by multiplying the loop variable by the lower- and upper bounds of the loop duration, i.e., [N * δᵢ, N * δₜ]. Note that since the LTC contains a preference over the number of loops, it implies a preference on the overall duration of the looping activity.

The looping temporal constraint can be viewed as a series of simple temporal constraints, but with the additional flexibility that the number of simple temporal constraints can be chosen from some range, and that there is a preference over how many are chosen (See Figure 2-2). While we restrict each loop to have the same controllable temporal bound, the same duration does not necessarily have to be chosen for each loop.
2.1.1 Specifying Search Patterns as Looping Activities

Since we focus on applications within search-and-rescue missions where coverage search patterns play a central role, we present examples of search patterns that may be modelled as looping activities. Figure 2-3 shows the iterations or "loops" within search patterns. Each pass of the lawnmower pattern can be represented as a loop, while the spiral pattern has loops of equal arc-length. For the zamboni and star patterns, each loop is shown in a different colour.

Specifying a controllable range of loops for each pattern allows us to model flexibility in the pattern. For example we could perform a broad search with few loops or a fine search with more loops (See Figure 2-4). The range of loops might be determined based on area size, sensor field-of-view and the vehicle flight altitude restrictions.

The temporal range per loop is determined based on vehicle dynamics and the distance to travel. For example, we can calculate the lower bound on the temporal duration by dividing the distance to travel by the maximum flight speed, and the upper bound by dividing the distance to travel by the minimum flight speed. For a lawnmower pattern, no matter how many we perform, the total side-wall distance (i.e. the sum of distance between loops) remains constant (See Figure 2-5). Thus the overall duration of a lawnmower pattern varies with the number of loops, but also
Figure 2-3: The loops in common search patterns.

Figure 2-4: Flexibility in the number of loops in a search pattern.

contains a fixed duration for the side-wall of the pattern. To model this, the temporal duration of a lawnmower pattern can be represented as a LTC for the loops in the pattern and a STC in series with it to represent the side-wall distance travelled. Associated with looping activities is also a preference function over the number of loops. For example, more loops performed in an area means we can collect more images or perform a finer resolution search, thus it should be preferred to performing
fewer loops.

The temporal durations of an Archimedes spiral, zamboni and star pattern can be modelled in a similar way to the lawnmower pattern as a LTC, or a combination of a LTC and a STC. The modelling of the spiral pattern is also shown in Figure 2-5.

2.2 Encoding Multiple Candidate Solutions in a Looping Temporal Plan Network

The input to our overall planning problem is a looping temporal plan network (LTPN); a compact encoding of many candidates for the optimal looping plan. A LTPN contains events representing points in time, looping episodes representing time-evolved looping activities and temporal constraints that specify durations between events. We encode choices as conditional decision variables, where an assignment to a decision variable activates a sub-plan in the LTPN, selecting a thread of execution. Episodes,
temporal constraints and decision variables are guarded, where guards are assignments to decision variables, and are only activated (part of the plan) when their guards are satisfied.

A looping episode (See Figure 2-6) is similar to a looping activity, but additionally contains a guard, to specify conditions under which that activity is valid. The scope of a looping episode is the start and end event that bounds it in the LTPN, as well as the conditional variables in its guard.

\[ \text{command}(p_1...p_n) \]

\[ \text{LTC : } \delta \in [\delta_l, \delta_u] \]
\[ N \in [N_l, N_u], \quad N \in \mathbb{Z}^+ \]
\[ f : N \rightarrow \mathbb{R}^+ \]

Figure 2-6: A general looping episode.

**Definition 3. Looping Episode:** A looping episode between events \( x_i \) and \( x_j \) is a triple \( < \text{command()}, \text{LTC}, G > \) where:

- \text{command()} is a method that can be directly interpreted by the robot and is of the form: \text{command}(p_1...p_n), where \( p_1 \) to \( p_n \) are parameters passed to the method.

- \text{LTC} is a looping temporal constraint between \( x_i \) and \( x_j \), as defined in Definition 2.

- \( G \) is a guard condition that activates the episode only if it holds true. A guard condition is an assignment to a decision variable (See Definition 4)

Looping episodes don’t need to explicitly contain all three attributes; for any episode it may contain \( \text{True} \) as a guard. If no temporal constraint is specified, a simple temporal constraint of \([0, \infty)\) is implied and simply means that the episode duration must be positive. As mentioned in Section 2.1, a simple temporal constraint
is a special case of a looping temporal constraint, where the number of loops is 1 and contains no preference. Using this special case, non-looping activities can also be modelled by a looping episode.

**Abbreviated Looping Episodes:**

<table>
<thead>
<tr>
<th>Activity</th>
<th>Start Time</th>
<th>Duration</th>
<th>Utility Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>search(A, star)</td>
<td>[0, 1]</td>
<td>[2, 3]</td>
<td>1.5 H</td>
</tr>
<tr>
<td>move(A, B)</td>
<td>[1, 2]</td>
<td>[3, 5]</td>
<td>10 log H</td>
</tr>
<tr>
<td>search(B, star)</td>
<td>[2, 3]</td>
<td>[4, 6]</td>
<td>2 H</td>
</tr>
<tr>
<td>move(A, C)</td>
<td>[3, 4]</td>
<td>[5, 7]</td>
<td>3 H</td>
</tr>
<tr>
<td>search(C, star)</td>
<td>[4, 5]</td>
<td>[6, 8]</td>
<td>3 log H</td>
</tr>
<tr>
<td>move(A, B)</td>
<td>[5, 6]</td>
<td>[7, 9]</td>
<td>3 H</td>
</tr>
<tr>
<td>search(B, lawnmm.)</td>
<td>[6, 7]</td>
<td>[8, 10]</td>
<td>2 H</td>
</tr>
<tr>
<td>move(A, C)</td>
<td>[7, 8]</td>
<td>[9, 11]</td>
<td>3 H</td>
</tr>
<tr>
<td>search(C, lawnmm.)</td>
<td>[8, 9]</td>
<td>[10, 12]</td>
<td>3 H</td>
</tr>
</tbody>
</table>

Figure 2-7: The single UAV search-and-rescue mission modelled as a LTPN. This LTPN encodes 8 different candidate solutions. Each looping episode represents a search pattern or a transition between areas.

Given the definition of a looping episode, we define a LTPN in Definition 4.

**Definition 4. Looping Temporal Plan Network (LTPN):** A looping temporal plan network is a tuple \( < X, DV, E, C > \) where:

- \( X \) is a set of temporal events. Two special events, \( (x_{start}, x_{end}) \) \( \in X \) represent the first and last events in the LTPN. Time assigned to \( x_{start} \) is always \( t = 0 \).
- **DV** is a set of conditional decision variables. Each decision variable \( dv \in DV \) has a discrete, finite domain \( I \), and may contain a guard condition that is an assignment to a decision variable.

- **E** is a set of looping episodes. Each looping episode \( ep \in E \) is bounded by a start and end event \( (x_i, x_j) \in X \) and is a tuple \( < \text{command}(), \text{LTC}, G > \) as defined in Definition 3.

- **C** is a set of conditional temporal constraints. Each \( c \in C \) specifies a duration between two events and may contain a guard condition that is an assignment to a decision variable.

Figure 2-7 shows a LTPN encoding a search-and-rescue mission. Each of the looping episodes represents a search pattern in an area, or a transition between areas.

### 2.3 Candidate Plan Generation

The looping temporal plan network in Figure 2-7 encodes 8 candidate solutions. Since those candidate solutions contain looping activities, the number of loops needs to be selected for each looping activity, all while maintaining temporal consistency. The first step is to generate a candidate plan by selecting threads of execution by making choices at decision variables. This section covers that problem.

Definition 5 defines a choice, while Figure 2-8 shows a visualization of a choice in a LTPN.

**Definition 5. Choice:** A choice in a LTPN \( < X, DV, C, E > \) is a pair \( < dv, i > \) where:

- \( dv \) is a decision variable i.e. \( dv \in DV \) with domain \( I \).
- \( i \) is a value in the domain of \( dv \), i.e. \( i \in I \).

Any episode, constraint or decision variable in the LTPN is active if its guard condition is satisfied. Since a choice is an assignment to a decision variable, making a
choice might satisfy guards that in turn activate episodes, temporal constraints and other decision variables. If a choice has been made for a decision variable, we say that the decision variable has been assigned. Figure 2-8 shows an assignment to the decision variable $A_E$ of $A_E = 1$ that activates the episode $move(A, lawnmower)$, the simple temporal constraint between events $S$ and $B_S$, and the decision variable $B_S$.

A complete set of choices (Definition 6) in a LTPN results in a complete candidate plan where all activated decision variables have been assigned, but the number of loops to be performed in each activity have not yet been calculated.

Definition 6. Complete Set of Choices: A complete set of choices in a LTPN is a set of assignments $A$ to decision variables such that:

- all decision variables activated by $A$ are assigned.
• all decision variables in \( A \) are either always active [29] or activated by a choice in \( A \).

Figure 2-9 shows a LTPN with a complete set of choices forming a complete candidate plan. Decision variable \( A_S \) is always active as it forms a part of all candidate plans. Because \( C_S \) is not activated by the assignment \( A_E = 1 \), it does not need to be assigned as part of the candidate plan.

### 2.4 Optimal Scheduling to produce a Looping Temporal Plan

We formulate the solution to a LTPN as a *looping temporal plan* (LTP or looping plan). A looping plan contains temporal events, executable looping activities and temporal constraints. An executable looping activity contains a command to be sent to the robot and a looping temporal constraint where an integer value from its loop range has been assigned to its loop variable (Figure 2-11). To allow temporal flexibility during the execution of the plan, the looping plan contains bounds on temporal constraints and loop durations rather than fixed values. Thus the difference between a candidate plan and a looping plan is that the latter contains a fixed number of loops in each activity. This section covers the problem of generating an optimal looping plan from a candidate plan. Figure 2-10 shows a candidate plan as an input and the associated looping plan as the output of the problem.

**Definition 7. Executable Looping Activity:** An executable looping activity between two temporal events, \( x_i \) and \( x_j \) is specified by a tuple \( <\text{command},LTC> \) where:

- \( \text{command} \) is a method that can be directly interpreted by the robot and is of the form: \( \text{command}(p_1..p_n) \), where \( p_1 \) to \( p_n \) are parameters passed to the method.

- \( LTC \) is a looping temporal constraint as defined in Definition 2, but where the loop variable has been fixed to a value in its loop range. The \( LTC \) specifies how many iterations to perform within the activity.
Candidate Plan:

Looping Plan:

Figure 2-10: A candidate plan and a looping plan. Note the number of loops $N$ is fixed as an integer in the looping plan. The temporal duration per loop is denoted by $\delta$, within a continuous range. Simple temporal constraints are shown as square brackets without a duration variable for simplicity.

![Diagram](image)

Figure 2-11: An executable looping activity contains a command to be sent to the robot and a looping temporal constraint where the number of loops has been fixed to a value in its loop range.

![Box](image)

**Definition 8. Looping Temporal Plan (LTP):** A looping temporal plan is a triple $< X, A, C >$ where:

- $X$ is a set of temporal events that designate points in time. Each $x \in X$ contains
a non-negative, real value. Two special events, \((x_{\text{start}}, x_{\text{end}}) \in X\) are distinct events that represent the first and last events in the LTP. \(x_{\text{start}}\) is always set to 0.

- \(A\) is a set of executable looping activities between events (Definition 7).

- \(C\) is a set of temporal constraints (looping temporal constraints or simple temporal constraints) that specify durations between events.

In order to execute a plan properly, activities need to be executed at times that satisfy temporal constraints in the looping plan. The dispatch time of an activity is the time that is assigned to its start event. The duration of an activity is the difference between the times assigned to its start and end event. The times at which activities in a looping plan are executed are specified as a schedule.

**Definition 9. Schedule for a LTP:** A consistent schedule \(\mathcal{T}\) for a looping temporal plan \(\mathcal{P}\) is a time assignment to each of its events \(X\), such that all temporal constraints and looping activity durations in \(\mathcal{P}\) are satisfied.

Note that in Definition 9 above, a looping activity duration is a function of the number of loops in that activity (set to an integer, \(N\)) and the temporal bound per loop (a real range, \([\delta_l, \delta_u]\)). Thus the duration of a looping activity between events \(e_i\) and \(e_j\), with time assignments \(t_i\) and \(t_j\) respectively, is satisfied if: \(N \times \delta_l \leq t_j - t_i \leq N \times \delta_u\).

Figure 2-12 shows a consistent, offline schedule for the looping plan presented in Figure 2-10.

To ensure the looping plan can be dispatched, we require that it is temporally consistent.

**Definition 10. Consistency of Looping Temporal Plans:** A looping temporal plan \(\mathcal{P}\) is consistent if a consistent schedule (Definition 9) exists.
2.5 Plan Reformulation to Dispatchable Form

Once the optimal looping plan has been found from the LTPN, it needs to be executed by a robotic system. We make use of an online dispatcher, to select times when to start and end activities in the plan. However, in order to do this, we need to compile the looping plan into a form without loops, called a dispatchable plan. A dispatchable plan is a least commitment temporal plan, where each looping activity in the looping plan has been instantiated as a series of sub-activities. While each executable looping activity contains a looping temporal constraint, each sub-activity contains a simple temporal constraint. All temporal bounds in the dispatchable plan need to be tightened to ensure a minimal network [10] (See Figure 2-13).

**Definition 11.** A sub-activity between two temporal events, $x_i$ and $x_j$ is specified by a tuple $<\text{command}(), \text{STC}>$ where:

- **command()** is a method that can be directly interpreted by the robot and is of the form: command($p_1...p_n$), where $p_1$ to $p_n$ are parameters passed to the method.

- **STC** is a simple temporal constraint, constraining the duration of the sub-activity between a lower and upper temporal bound: $[lb, ub]$.

Given the definition of a sub-activity above, we define a dispatchable temporal plan; the final output from our planning executive:
Figure 2-13: A looping plan and a dispatchable temporal plan. Note that the dispatchable plan contains a series of sub-activities rather than a looping activity and it contains tightened temporal bounds and temporal constraints.

**Definition 12. Dispatchable Temporal Plan:** A dispatchable temporal plan is a triple $< X, A, C >$ where:

- $X$ is a set of temporal events that designate points in time. Each $x \in X$ contains a non-negative, real value. Two special events, $(x_{start}, x_{end}) \in X$ are distinct events that represent the first and last events in the plan. $x_{start}$ is always set to 0.

- $A$ is a set of sub-activities between events (Definition 11).

- $C$ is a set of simple temporal constraints that specify durations between events.

Note that in the definition above, all sub-activity durations and temporal constraints are tightened to a minimal network.
2.6 Chapter Summary

This thesis attempts to solve the problem of optimal hierarchical planning over looping activities. The goal is to optimize overall plan utility and provide a dispatchable plan to a robot. We showed how many candidate solutions to a search-and-rescue problem may be modelled as a looping temporal plan network (LTPN); the input of our problem.

We presented the solution to a LTPN as a looping plan (LTP), with a complete set of assignments to decision variables, integer assignments to looping variables and temporally consistent bounds on loop temporal bounds and temporal constraints. We then presented the output of the overall problem as a dispatchable plan, containing bounds on durations rather than a fixed schedule in order to be dispatched on-line and deal with temporal disturbances.
Chapter 3

Optimal Scheduling of Looping Activities

In this chapter, we describe our capability for the optimal scheduling of a looping temporal problem with preference (previously published in [22]). If we assume that a planner has made choices within a temporal plan network with loops to form a candidate plan, the problem of finding the optimal looping plan corresponds to solving a looping temporal problem with preference (LTPP).

In Section 3.1 we present an example problem similar to the one in Chapter 1, but without choice. Next, we define the looping temporal problem with preference (LTPP) in Section 3.2 and present the state of the art candidate solution methods in Section 3.3. Then we present our approach to solving the LTPP in Section 3.4 and demonstrate it on the example problem as a means to autonomously choose the number of loops within search patterns, while adhering to temporal constraints. Finally, we conclude the chapter in Section 3.5.

3.1 Autonomous Scouts for Search-and-Rescue

As an illustrative example of a scouting mission with looping activities and to help explain our algorithm for solving LTPPs, we re-visit the search and rescue example introduced in Section 1.1. Suppose that a hierarchical planner has selected a lawn-
mower search pattern for searching Area A, a star search pattern for Area B, and path-following on the path between the two areas. The lawnmower pattern can be represented as a looping activity with each pass representing a loop, while the star pattern is a looping activity with each triangle as a loop (see Section 2.1).

Given the dynamics of the UAV and the size of the areas, each loop in lawnmower pattern can be performed in a controllable duration between 2 and 5 minutes and each loop in the star pattern between 3 and 4 minutes. The length of the path between Area A and B can be flown in 3 to 7 minutes. Given flight altitude restrictions and the sensor's field-of-view on board, the system determines that the UAV has to perform between 5 and 20 loops in Area A, and at least 5 loops in Area B. Based on prior beliefs and expected information gain, there is also a preference over the number of loops to perform in each area. These are $f_A(N_A) = 10 \log (N_A)$ for the lawnmower pattern and $f_B(N_B) = 2N_B$ for the star pattern, where $N$ denotes the number of loops. Since the number of loops cannot be fractional, these functions are discrete. A flight corridor between Area A and B opens briefly between 25 and 35 minutes after the start of the mission, allowing transition from Area A to Area B. Finally, the UAV has a maximum flight time of 50 minutes. Figure 3-1 summarizes this information.

This problem can be encoded as a LTPP. Looping activities, constrained by time, are represented by looping temporal constraints (LTCs) in the LTPP. Path-following and time constraints such as the overall mission duration and reaching Area B between 25 and 35 minutes can also be represented as a looping activity with only one loop. The preference functions map the numbers of loops in each LTC to a utility value and a global preference function calculated the overall utility of the mission.

The optimal solution to this problem would be to perform 11 loops in Area A and 8 loops in Area B.
3.2 Defining the Looping Temporal Problem with Preference (LTPP)

We define a new class of problem called the looping temporal problem with preference (LTPP) as an encoding of a problem containing temporal activities with loops. Our encoding contains the minimal variables and constraints that are sufficient to determine the number of loops and durations. Using our encoding, a looping program can be mapped to an LTPP in a straightforward manner.

Definition 13. Looping Temporal Problem with Preference (LTPP): A LTPP is a triple < X, C, F > where:

- X is a set of events representing points in time.
- C is a set of looping temporal constraints (LTCs) between pairs of events as defined in Definition 14.
• \( F \) is a function that maps local utility values from the LTCs to a global utility value.

**Definition 14. Looping Temporal Constraint:** (Previously defined in Chapter 2)
A looping temporal constraint (LTC) is a simple temporal constraint repeated between two time events, \( x_i \) and \( x_j \), and is specified by a tuple \(< N_{ij}, \delta, f >\) where:

- \( \delta \) is the duration of a simple temporal constraint between the start and end times of one loop, and is of the form \( \delta \in [\delta_l, \delta_u], \delta \in \mathbb{R}^+ \). This is known as the **loop duration**.

- \( N \) is an integer **loop variable** between the events \( x_i \) and \( x_j \) specifying how many loops to execute. The loop variable is constrained in a **loop range**: \( N \in [N_l, N_u], N \in \mathbb{Z}^+ \).

- \( f \) is a monotonic preference function that maps the number of loops \( N \) to a real utility value, \( f : N \rightarrow \mathbb{R}^+ \).

The solution method proposed in this paper restricts the global preference function \( F \) to a monotonic function (that may be non-convex), i.e. when given any range over the number of loops, the preferred solution to each loop variable is always at one extreme of its range. In the experimentation section, we focus on one class of preference functions that exhibits this property. In this class of preferences, the local functions are monotonic and produce positive, real numbers. The global preference function then combines the local preferences using any combination of the + and \( \times \) operators.

As mentioned in Chapter 2, a simple temporal constraint is a special case of a looping temporal constraint, where the number of loops is 1. As these can be easily extracted from the set of all looping temporal constraints \( C \), we will refer the set of simple temporal constraints as \( STC \subseteq C \) and the set of remaining looping temporal constraints as \( LTC \subseteq C \). Thus, the LTPP consists of hard temporal constraints (\( STC \)) that have to hold and soft temporal constraints (\( LTC \)) with preference functions over them (Figure 3-2).
LTC: \( i \xrightarrow{} j \)

\( N_{ij} \in [N_{ijl}, N_{iju}], N_{ij} \in \mathbb{Z}^+ \)

\( \delta_{ij} \in [\delta_{ijl}, \delta_{iju}] \)

\( f_{ij} : N_{ij} \rightarrow \mathbb{R}^+ \)

STC: \( i \xrightarrow{} j \)

\( \delta_{ij} \in [\delta_{ijl}, \delta_{iju}] \)

Figure 3-2: The LTPP is comprised of looping temporal constraints, events and a global utility function. Simple temporal constraints are a special case of looping constraints, but are called out explicitly for ease of understanding.

Given these definitions, the search-and-rescue example is framed as a LTPP in Figure 3-3. The global preference function \( F \) combines the two local preference functions using the + operator, i.e. \( F = (10 \log(N_A) + 2N_B) \).

Figure 3-3: The search-and-rescue mission framed as a LTPP.

Note that loops are atomic, and the system can not arbitrarily exit a loop in its middle. Consequently, the loop variables are restricted to be integer and thus a looping temporal constraint \( (LTC_{ij}) \) between two time events \( x_i \) and \( x_j \) can not simply be written as \( x_j - x_i \in [N_{ij} \times \delta_{ij}, N_{uij} \times \delta_{uij}] \), since the integer constraint on
$N_{ij}$ causes a disjunctive temporal bound between $x_i$ and $x_j$. For example the LTC:

$$\{N_{ij} \in [1, 2]; \delta_{ij} \in [3, 4]\} \equiv \{[3, 4] \lor [6, 8]\}$$

has a temporal gap between 4 and 6.

An **optimal solution** to a LTPP is an integer assignment to each LTC looping variable $N$, that maximizes the global preference function, while satisfying all temporal constraints.

### 3.3 Candidate Solution Methods

Candidate solution methods exist in the scheduling and mathematical optimization communities for solving a looping temporal problem with preference (LTPP). These are described in the sections below, along with their limitations for solving our specific problem.

#### 3.3.1 Scheduling Community

First, for loops with finite bounds, the loops in a LTPP can be encoded in a temporal constraint satisfaction problem with preference (TCSPP) [23] or a disjunctive temporal problem with preference (DTPP) [23][25], where each loop forms a disjunctive temporal bound between two events and the preference function over the number loops can be split into one function for each of the disjunctive bounds. For example, the lawnmower pattern in Area A of the search-and-rescue example could be framed as in Figure 3-4. However, a LTPP is inefficient to solve in the TCSPP framework as TCSPP solvers break the disjunctive constraints into component simple temporal problems (STPs)[10] to solve. Compared to a TCSPP or DTPP, the LTPP also allows a clean way of specifying looping ranges such as “anything more than”, by setting infinity as an upper bound (such as the pattern in Area B of the original search-and-rescue mission example).
3.3.2 Mathematical Optimization Community

The LTPP contains a mixture of integer and real-valued constraints, as well as a maximization over non-linear, non-convex functions. Thus, the second candidate solution method is to frame the problem as a mixed integer non-linear program (MINLP). MINLPs are among the class of theoretically difficult problems that are NP-complete. Solution methods for solving MINLPs include outer approximation methods ([1], [13]), branch-and-bound (B&B) ([3], [24]), generalized benders decomposition [15] and extended cutting plane methods [26]. These approaches generally rely on the successive solutions of closely related non-linear program (NLP) problems. In particular, B&B starts out forming a pure continuous NLP problem by relaxing the integer constraints on the discrete variables.

Section 3.2 defined the LTPP as a triple $< X, C, F >$. The problem can be framed as a mixed integer non-linear program as problem contains integer and real constraints, and the preference functions may be non-linear.

The MINLP takes the following general form:

$$\text{maximize} : f(x, y)$$

subject to: $g(x, y) \leq 0$

$x \in X$

$y \in Y \text{ integer}$

In the case of the LTPP, the objective function tries to maximize some combination of the LTC preference functions as follows:
maximize: $F\left(\{N_{ij}\}\right) \forall i, j$

Then, each looping temporal constraint ($LTC_{ij} \in C$), between events $x_i, x_j \in X$, with $N_{ij} \in [N_{ijl}, N_{iju}]$ and $\delta_{ij} \in [\delta_{ijl}, \delta_{iju}]$ can be expressed as the following constraints:

\[
N_{ij}\delta_{ijl} \leq x_j - x_i \leq N_{ij}\delta_{iju}
\]

\[
N_{ijl} \leq N_{ij} \leq N_{iju}
\]

\[
N_{ij} \in \mathbb{Z}^+
\]

\[
\forall i, j
\]

which can then be written as linear constraints:

\[
x_j - x_i \leq N_{ij}\delta_{iju}
\]

\[
x_i - x_j \leq -N_{ij}\delta_{ijl}
\]

\[
N_{ij} \leq N_{iju}
\]

\[
- N_{ij} \leq -N_{ijl}
\]

\[
N_{ij} \in \mathbb{Z}^+
\]

Thus, the LTPP can compiled to a MINLP as presented in Figure 3-5.

\[
\text{maximize : } F\left(\{N_{ij}\}\right)
\]

\[
\text{subject to : } x_j - x_i \leq N_{ij}\delta_{iju}
\]

\[
x_i - x_j \leq -N_{ij}\delta_{ijl}
\]

\[
N_{ij} \leq N_{iju}
\]

\[
- N_{ij} \leq -N_{ijl}
\]

\[
N_{ij} \in \mathbb{Z}^+
\]

\[
\forall i, j
\]

Figure 3-5: The MINLP encoding of a LTPP.
3.4 Our Approach to Solving the LTPP

Drawing from techniques used by TCSPP, DTPP and B&B MINLP solvers, our approach solves the LTPP by casting the original problem as a series of simple temporal problems (STPs), relaxing the integrality requirements at each step and checking consistency. Where our approach differs from current scheduling techniques, is that it can handle infinite loops, by employing a domain filtering technique to prune loop ranges and then it searches through the remaining state-space by checking consistency of STPs that encompass *ranges* of loops rather than a single loop value combination. The consistency checks are performed incrementally and a tight heuristic is used to minimise changes between consistency checks.

We present the domain filtering algorithm, **DOMAINFILTER** in Section 3.4.1 and the search algorithm **BOUNDSEARCH** in Section 3.4.2. Note that we will refer to the combination of the these two algorithms in future chapters simply as **SOLVE_LTPP**.

### 3.4.1 Domain Filtering

The **DOMAINFILTER** algorithm (Algorithm 1) reduces the solution state-space by pruning out loop ranges that can never form part of a solution. In the example presented earlier, one of the loop ranges can take on a value between 5 and ∞. However, an infinite number of loops cannot form part of the solution as the mission is constrained to take less than 50 minutes. Reducing the loop ranges to a space of possible solutions greatly reduces the search space that the **BOUNDSEARCH** algorithm (Algorithm 2) has to search through. Figure 3-6 shows the input and output of **DOMAINFILTER** running on the example search-and-rescue problem.

**DOMAINFILTER** extends the all-pairs shortest path method to readily prune loops in the LTPP at each iteration. For simplicity, the pseudo code for the extension to the Floyd-Warshall algorithm [14] is shown in Algorithm 1, but the implementation for benchmarking uses an extension to Johnson's algorithm [17].
DOMAINFILTER begins by forming a STP from a LTPP (line 1), by relaxing the integer constraint on the loop ranges, thus forming a STC from each LTC as follows:

\[
LTC : N \in [N_l, N_u]; \delta \in [\delta_l, \delta_u], \quad N \in \mathbb{Z}^+; \delta \in \mathbb{R}^+ \\
\Rightarrow STC : d \in [N_l \times \delta_l, N_u \times \delta_u], \quad d \in \mathbb{R}^+
\]

If the STC between event \( x_1 \) and event \( x_2 \) has the bound \( d \in [d_l, d_u] \), then \( d(x_1, x_2) \) is the distance edge from \( x_1 \) to \( x_2 \) (equal to \( d_u \)) and \( d(x_2, x_1) \) is the distance edge from \( x_2 \) to \( x_1 \) (equal to \(-d_l\)).

The algorithm runs exactly the same as the Floyd-Warshall algorithm until line 7. At this point, if the algorithm tightens a bound on the STC (line 6), the loop range of the corresponding LTC may be tightened too as any loop number whose corresponding temporal bound falls outside the tightened STC bound is infeasible and can be pruned (See Figure 3-7). This is done by the `pruneLoops()` function in line 7. The `pruneLoops()` function accepts the tightened STC bound \( d(x_i, x_j) \) and the cor-
Algorithm 1: DomainFilter

Output: A LTPP, with loop ranges tightened to remove inconsistent solutions.

Algorithm
1. STP ← makeSTP(LTPP)
2. for x_k ∈ X do
3.     for x_i ∈ X do
4.         for x_j ∈ X do
5.             if (d(x_i, x_k) + d(x_k, x_j)) < d(x_i, x_j) then
6.                 d(x_i, x_j) ← (d(x_i, x_k) + d(x_k, x_j))
7.                 C_ij ← pruneLoops(C_ij, d(x_i, x_j))
8.                 d(x_i, x_j) ← reTighten(d(x_i, x_j), C_ij)
9. for x_i ∈ X do
10.    if d(x_i, x_i) < 0 then
11.        return nil
12. return LTPP

responding LTC, C_ij with N ∈ [N_l, N_u] and δ ∈ [δ_l, δ_u]. If d(x_i, x_j) is a lower-bound in the STC, it tightens loop ranges as in the left equation below, and if d(x_i, x_j) is an upper-bound, it tightens as in the right equation:

\[ N_l \geq \text{ceil}\left(\frac{d(x_i, x_j)}{\delta_u}\right) \quad \text{and} \quad N_u \leq \text{floor}\left(\frac{d(x_i, x_j)}{\delta_l}\right). \]

The floor() and ceil() operators ensure the loop range bounds remain integer, but since these operator further tighten the loop range bounds, the relaxed STC bound d(x_i, x_j) could be tightened once more. This is done by reTighten() (line 8) by tightening it to max(d(x_i, x_j), N_l × δ_l) or min(d(x_i, x_j), N_u × δ_u), if d(x_i, x_j) is a lower- or upper-bound respectively.

The loop pruning process may be more clearly understood by noting that since \( \delta \in [\delta_l, \delta_u] \) is a controllable range, the largest number of loops that would fall within the tightened temporal bound \( d \in [d_l, d_u] \), would be when \( N \times \delta_l \leq d_u \), and the smallest number of loops within the range \( d \) would be \( N \times \delta_u \geq d_l \).
If the network of the STP contains a negative cycle (line 10), we return nil (line 10) as it is inconsistent and thus the original LTPP is also inconsistent as each STC temporal range encompassing the entire corresponding LTC temporal range. The converse is not necessarily true; if the relaxed STP network of the LTPP is consistent, no solution to the LTPP is guaranteed. The relaxation of the integer constraint on the loop variables to form a STP from a LTPP encompasses all looping temporal ranges, but also adds temporal ranges that lie between the loop temporal ranges and that don’t exist in the original LTPP. Figure 3-8 shows an example of an inconsistent solution after running DOMAINFILTER, where the range of loops for each LTC is tightened from [1, 5] to [1, 2]. Although the network of the relaxed STP is consistent and the loops ranges have been pruned, no solution exists to the LTPP, since there is a requirement of 4.5 for the overall duration, whereas the duration of every looping constraint needs to be an integer of exactly 1 or 2 in this case.

The Johnson’s Algorithm version of DOMAINFILTER works similar to what has been explained above; every time a distance edge in the relaxed problem is tightened, the pruneLoops() and reTighten() methods are performed. The run-time complexity of the Johnson’s version of DOMAINFILTER is $O(X^2 \log C + XC)$.

Running DOMAINFILTER on the search-and-rescue example, results in a large reduction domain of looping ranges. The loop range of the lawnmower pattern is reduced from [5, 20] to [5, 16] and the loop range of the star pattern is reduced from [5, ∞) to [5, 8] (see Figure 3-6).
3.4.2 Searching for the Optimal Solution

After running DomainFilter to reduce the search-space, BOUNDSEARCH (Algorithm 2) takes a best-first approach to search through the remaining space of loop ranges for an optimal, integer assignment to the looping variables, given the global preference function. It explores the search space by incrementally checking consistency over ranges of possible loop combinations, pruning large spaces of inconsistent combinations. When loop range combinations are consistent, our algorithm splits them into narrower combinations until an integer solution is found. It uses an admissible heuristic to guide the search and expands the search tree in best-first order. Thus, the first consistent, integer assignment to the LTPP loop ranges is guaranteed to be the optimal solution.

The search tree for the first three iterations of the algorithm running on the search-and-rescue problem (Figure 3-9) is used to help explain BOUNDSEARCH.

Candidates:

BOUNDSEARCH makes use of candidates (not to be confused with candidate plans) to maintain LTC loop ranges we are checking consistency over, along with some other information need during the search. Each candidate, $D$ is a node in the search tree and is a tuple $< L, H, P, C >$ that maintains the following information:
Algorithm 2: \texttt{BOUNDSEARCH}

\textbf{Input:} A LTPP with a LTC loop ranges pruned by \texttt{DOMAINFILTER}.

\textbf{Output:} A LTPP schedule, with integer assignments to each LTC loop range.

\textbf{Initialization:}
1. \( D \leftarrow \text{makeCandidate(LTPP)} \)
2. \( \text{priorityQ} \leftarrow \{D\} \)
3. \( \text{STP} \leftarrow \text{makeSTP(LTPP)} \)
4. \( \text{ITC} \leftarrow \text{initializeITC(STP)} \)

\textbf{Algorithm:}
5. \textbf{while} \( \text{priorityQ} \neq \emptyset \) \textbf{do}
6. \quad \( D \leftarrow \text{pop(priorityQ)} \)
7. \quad \( C \leftarrow \text{getModifiedConstraint}(D) \)
8. \quad \textbf{if} \( \text{consistentITC?(ITC, C)} \) \textbf{then}
9. \quad \quad \textbf{if} \( \text{integerAssignment?(D)} \) \textbf{then}
10. \quad \quad \quad \textbf{return} \( \text{makeSolution(LTPP, D)} \)
11. \quad \quad \textbf{else}
12. \quad \quad \quad \{\text{children,} \ C\} \leftarrow \text{split}(D) \)
13. \quad \quad \quad \textbf{for} \text{child in children} \textbf{do}
14. \quad \quad \quad \quad \text{setModifiedConstraint(child,} \ C) \)
15. \quad \quad \quad \quad \text{setParent(child,} \ D) \)
16. \quad \quad \quad \quad \text{addToQ(priorityQ,} \ \{\text{children}\}) \)
17. \quad \quad \textbf{else}
18. \quad \quad \quad \text{P} \leftarrow \text{parent(peek(priorityQ))} \)
19. \quad \quad \quad \text{resetITC(P)} \)
20. \textbf{return} \( \text{nil} \)

- \( L \): a set of LTC loop ranges to check consistency over.

- \( H \): an admissible heuristic value used to guide the search and is equal to the maximum utility obtainable from \( L \).

- \( P \): the parent candidate of \( D \).

- \( C \): the LTC that was split to produce \( D \) from \( P \).

To explain the above, consider the search tree in Figure 3-9. The set of LTC loop ranges to check consistency over correspond to the ranges \( N_A \) and \( N_B \). \( D_1 \) is the parent of \( D_2 \) and \( D_3 \). The constraint that was split to produce \( D_2 \) and \( D_3 \) from \( D_1 \) is the LTC corresponding to the range \( N_A : [5,16] \). Since the global preference
function $F$ is monotonically increasing, the maximum utility $H$ for any candidate is found by simply evaluating $F$ at the maximum loop range for each $L_i \in L$. In the example (Figure 3-9), $H_{D1} = f_A(16) + f_B(8) = 44$.

Figure 3-9: The search tree after three iterations of the search-and-rescue problem. Looping ranges are marked $N_A$ and $N_B$ to denote the two search areas. Candidates are labelled in the order they are added to the queue and the solution is shown as a double circle node.

Initialization:

**BOUNDSEARCH** begins with an initial candidate on the queue. The initial candidate ($D1$ in Figure 3-9) contains the entire search-space. The Incremental Consistency Checker *ITC* is initialized (line 4) with a STP that is formed from the LTPP by relaxing all integer constraints on the looping variables (line 3). For the search-and-rescue example, *ITC* would be initialized to the STP corresponding to $D1$ in Figure 3-10.

Algorithm:
Within each iteration, **BOUNDSEARCH** removes the best candidate $D$ from the queue (according to $H$) (line 6). If $D$ is found to be consistent, it is split into two children candidates that each contain a subset of the state-space of possible loop combinations (line 11). In Figure 3-9, $D_1$ is found to be consistent, and is split into two candidates, $D_2$ and $D_3$. Only one loop interval is split to ensure the two children encapsulate all the possible loop combinations of the parent. The LTC loop interval that adds the most utility to the overall preference function is chosen as the one to split, in order to guide the search to the optimal solution faster.

In the process of splitting from a parent candidate, only one constraint $C$ is modified to form each child (see Figure 3-10) and for each child, this constraint is saved (lines 11-13). Because a tight heuristic is used, we know that one of the children will be the next-best candidate on the queue and thus when it is popped off the queue, we check consistency incrementally by simply modifying $ITC$ with the single constraint $C$.

The algorithm continues splitting candidates and narrowing the possible loop combinations of each candidate. If an inconsistent candidate is found ($D_6$), it is pruned, removing all its possible loop combinations from the search space. At this stage $ITC$ is in an inconsistent state and it is reset to the consistent state it was in for the parent ($D_4$) of the next-best candidate on the queue ($D_7$) (line 17). Then, at the next iteration of the algorithm, consistency is checked by simply modifying $ITC$ again with the single constraint $C$ that was altered when $D_7$ was formed from $D_4$.

Figure 3-10 shows how only one constraint is modified while splitting a parent candidate into two children.

If **BOUNDSEARCH** keeps splitting loop ranges until an integer loop combination is found (i.e. the lower and upper bound of each loop range is the same), it assigns the candidate loop values to the LTPP, before returning it as the optimal result (line 10). If **BOUNDSEARCH** loops until the queue is empty, no solution is possible to the LTPP and **nil** is returned (line 18).

The output of the algorithm is an integer assignment to the loop variables (i.e. the lower- and upper-bound of the loop range is the same) that maximizes utility, while
Figure 3-10: STP representation of candidates after the first split. The loop range that adds the most to the overall utility is [5, 16] and it is split to [5, 10] and [11, 16]. Since each loop has a temporal duration of \( \delta_B \in [2, 5] \), STCs of \([5 \times 2, 10 \times 5]\) and \([11 \times 2, 16 \times 5]\) are formed.

respecting all temporal constraints. The output from the search-and-rescue problem is shown in Figure 3-11.

Note that the commands associated with looping activities are ignored when framing the problem as a LTPP. However, after solving the LTPP, if we add commands back to the looping temporal constraints, we form executable looping activities (see Definition 7) and thus a looping temporal plan (Definition 8).

**Guarantee on temporal consistency:**
The output of BOUNDSEARCH contains an integer assignment to each LTC loop variables. Given an integer assignment to a LTC between two events \( x_i \) and \( x_j \): \( \{N = c; \delta \in [\delta_l, \delta_u]\} \), we can write the overall temporal bound as \([N \times \delta_l, N \times \delta_u]\) as the integer value of \( N \) no longer causes a disjunctive temporal bound between \( x_i \) and \( x_j \) (note that this was not the case when \( N \) was a value within a discrete range).

Thus consistency of the relaxed STP problem implies consistency of the LTPP.
3.5 Chapter Summary

In this chapter we presented the looping temporal problem with preference (LTPP), as a formalism for framing scheduling problems that contain looping activities and preferences over them. We then presented a scheduling algorithm that finds the number of loops for each looping activity to maximize a global preference function of the looping variables, while respecting all temporal constraints.

The next chapter describes planning with loops. The algorithm for solving the planning problem calls the LTPP solver as a sub-routine, to calculate the optimal looping plan from a candidate plan.
Chapter 4

Planning over Looping Activities

The capability we aim to provide in this thesis is dynamic execution of a looping temporal plan network (LTPN), focusing on the problem of mapping a LTPN to the optimal dispatchable temporal plan (without loops). Chapter 3 introduced and solved the preferred scheduling problem, the core to the overall planning system (see Figure 4-1). To complete the planning system, this chapter focuses on the candidate generation problem, optimality checking and the reformulation of a looping plan to a dispatchable temporal plan.

The problem is framed within a looping temporal plan network (LTPN). Our goal is first to find the optimal looping plan in the LTPN by making assignments to decision variables and loop variables of each looping activity, and second to reformulate the looping plan into a dispatchable temporal plan. While we can guarantee that a schedule exists in the dispatchable plan before returning it, we return a least-commitment, minimal network to allow more flexibility during online execution.

The structure of this chapter is as follows. First, we present an example in Section 4.1, that we use to explain the problem and our approach. In Section 4.2, we model the problem within a looping temporal plan network (LTPN). Previous work we build upon is presented in Section 4.3. Next, we present our approach for finding the optimal looping plan in Section 4.4 and our approach for reformulating the looping plan into dispatchable from in Section 4.5. We summarize this chapter in Section 4.6.
Figure 4-1: The planning system architecture. Dispatching is outside the scope of this thesis as it has been completed in prior work. Optimal scheduling (Part 2) is covered in Chapter 3.
4.1 Clarifying Example

In this section we extend the problem introduced in Chapter 1, thus if the reader is unfamiliar with the problem, we suggest they review it in Section 1.1.

The following requirements are placed on the mission:

1. Search Area A.

2. Search Area B or Search Area C.

3. Search the path between Area A and the area chosen in Requirement 2.

4. Complete the mission within 50 minutes (battery life limit).

5. If Area B is chosen in Requirement 2, transition there between 25 and 35 minutes after the start of the mission (flight corridor is blocked outside of this time).

We assume we have an automated system that determines suitable search pattern types, search pattern loop ranges and temporal bounds per loop from information including geography, search area sizes and shapes, vehicle dynamics and the sensor field-of-view. Based on the prior belief of where the hikers are, the system also calculates the utility as a function of loops performed in each search pattern. The search pattern types and parameter options to search each area are shown in Figure 4-2.

Similar to Chapter 3, $\delta$ denotes the temporal bound per loop, $N$ denotes the number of loops to perform and $f(N)$ denotes the utility function over the number of loops. We denote the duration variable of a simple temporal constraint by $\delta$ as well as a simple temporal constraint is a looping temporal constraint with one loop and no preference.

If the utility functions are calculated according to expected information gain from a prior belief [6], then maximizing the overall utility of the mission within the imposed temporal constraints corresponds to the best chance of finding the hikers. This corresponds to making choices as to which areas to search, which search patterns
Search Pattern Options and Parameters:

<table>
<thead>
<tr>
<th>Area A:</th>
<th>Area B:</th>
<th>Area C:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{search}(A, \text{star})$</td>
<td>$\text{search}(B, \text{star})$</td>
<td>$\text{search}(C, \text{star})$</td>
</tr>
<tr>
<td>$\delta_{A1} \in [2, 3]$</td>
<td>$\delta_{B1} \in [3, 4]$</td>
<td>$\delta_{C1} \in [3, 4]$</td>
</tr>
<tr>
<td>$N_{A1} \in [5, 10]$</td>
<td>$N_{B1} \in [5, 10]$</td>
<td>$N_{C1} \in [5, 20]$</td>
</tr>
<tr>
<td>$f(N_{A1}) = 1.5N_{A1}$</td>
<td>$f(N_{B1}) = 2N_{B1}$</td>
<td>$f(N_{C1}) = 3\log(N_{C1})$</td>
</tr>
<tr>
<td>$\text{search}(A, \text{lawnm.})$</td>
<td>$\text{search}(B, \text{lawnm.})$</td>
<td>$\text{search}(C, \text{lawnm.})$</td>
</tr>
<tr>
<td>$\delta_{A2} \in [2, 5]$</td>
<td>$\delta_{B2} \in [5, 7]$</td>
<td>$\delta_{C2} \in [2, 6]$</td>
</tr>
<tr>
<td>$N_{A2} \in [5, 20]$</td>
<td>$N_{B2} \in [5, 8]$</td>
<td>$N_{C2} \in [5, 10]$</td>
</tr>
<tr>
<td>$f(N_{A2}) = 10\log(N_{A2})$</td>
<td>$f(N_{B2}) = 3N_{B2}$</td>
<td>$f(N_{C2}) = N_{C2}$</td>
</tr>
</tbody>
</table>

Legend:
- $N$: Number of loops
- $\delta$: Loop duration
- $f$: Utility function

Figure 4-2: Search-and-rescue planning example.
to perform and how many loops to perform within each pattern, while respecting
temporal constraints.

4.2 Modelling the Problem

The problem may be modeled by looping episodes, temporal constraints, events and
decision variables in a LTPN. Figure 4-4 shows the LTPN for the example search-
and-rescue mission.

Figure 4-3 shows the general form of a looping episode, first presented in Chapter 2.
All search patterns presented in the example may be modeled as a looping episodes.

<Guard>
\text{command}(p_1...p_n)

\text{LTC} : \delta \in [\delta_l, \delta_u]
N \in [N_l, N_u], N \in \mathbb{Z}^+
f : N \rightarrow \mathbb{R}^+

Figure 4-3: A general looping episode

The activities of moving between areas may also be modeled as a looping episode
with a single loop.

Furthermore, choices such as search pattern type and which area to move to from
Area A may be modeled as decision variables. The decision variables in the example
include selecting a search pattern for Area A, B and C (Variables $A_S$, $B_S$ and $C_S$),
and selecting whether to move to Area B or C from Area A (Variable $A_E$). Each
of these variables have domain \{1, 2\} as each variable has two possible threads of
execution that it can activate.

The temporal constraint on when to move to Area B is conditional on whether
Area B is selected to be searched or not. This can be modeled as a conditional simple
temporal constraint, that has a guard condition of $A_E = 1$ associated with it.
Finally, start and end events mark the start and end of the mission, and an unconditional simple temporal constraint between the start and end events restricts the mission to last less than 50 minutes.

Abbreviated Looping Episodes:

<table>
<thead>
<tr>
<th>search(A, star)</th>
<th>move(A, B)</th>
<th>search(B, star)</th>
<th>search(C, star)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt; A_E = 1 &gt;)</td>
<td>(&lt; A_E = 1 &gt;)</td>
<td>(&lt; B_E = 1 &gt;)</td>
<td>(&lt; C_E = 1 &gt;)</td>
</tr>
<tr>
<td>(\delta_{A1} \in [2,3])</td>
<td>(\delta_{AB} \in [3,7])</td>
<td>(\delta_{B1} \in [3,4])</td>
<td>(\delta_{C1} \in [3,4])</td>
</tr>
<tr>
<td>(N_{A1} \in [5,10])</td>
<td>(f(N_{A1}) = 1.5N_{A1})</td>
<td>(N_{B1} \in [5,10])</td>
<td>(f(N_{B1}) = 2N_{B1})</td>
</tr>
<tr>
<td>search(A, lawnm.)</td>
<td>move(A, C)</td>
<td>search(B, lawnm.)</td>
<td>search(C, lawnm.)</td>
</tr>
<tr>
<td>(&lt; A_S = 2 &gt;)</td>
<td>(&lt; A_S = 2 &gt;)</td>
<td>(&lt; B_S = 2 &gt;)</td>
<td>(&lt; C_S = 2 &gt;)</td>
</tr>
<tr>
<td>(\delta_{A2} \in [2,5])</td>
<td>(\delta_{AC} \in [5,9])</td>
<td>(\delta_{B2} \in [5,7])</td>
<td>(\delta_{C2} \in [2,6])</td>
</tr>
<tr>
<td>(N_{A2} \in [5,20])</td>
<td>(f(N_{A2}) = 10\log(N_{A2}))</td>
<td>(N_{B2} \in [5,8])</td>
<td>(f(N_{B2}) = 3N_{B2})</td>
</tr>
<tr>
<td>search(A, lawnm.)</td>
<td>move(A, C)</td>
<td>search(B, lawnm.)</td>
<td>search(C, lawnm.)</td>
</tr>
<tr>
<td>(&lt; A_S = 2 &gt;)</td>
<td>(&lt; A_S = 2 &gt;)</td>
<td>(&lt; B_S = 2 &gt;)</td>
<td>(&lt; C_S = 2 &gt;)</td>
</tr>
<tr>
<td>(\delta_{A2} \in [2,5])</td>
<td>(\delta_{AC} \in [5,9])</td>
<td>(\delta_{B2} \in [5,7])</td>
<td>(\delta_{C2} \in [2,6])</td>
</tr>
<tr>
<td>(N_{A2} \in [5,20])</td>
<td>(f(N_{A2}) = 10\log(N_{A2}))</td>
<td>(N_{B2} \in [5,8])</td>
<td>(f(N_{B2}) = 3N_{B2})</td>
</tr>
<tr>
<td>f(N_{A1}) = 1.5N_{A1}</td>
<td>f(N_{B1}) = 2N_{B1}</td>
<td>f(N_{C1}) = 3\log(N_{C1})</td>
<td></td>
</tr>
<tr>
<td>f(N_{A2}) = 10\log(N_{A2})</td>
<td>f(N_{B2}) = 3N_{B2}</td>
<td>f(N_{C2}) = N_{C2}</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4-4: Single UAV search mission modelled as a LTPN. This LTPN compactly encodes 8 candidate solutions.

Given this LTPN, the problem is then to make temporally consistent assignments to decision variables and loop variables in order to maximize utility, and then the compile the looping plan into dispatchable form. The result is a least-commitment temporal plan that can be executed by the Pike executive [20], where the looping temporal constraints have been instantiated as a series of simple temporal constraints.
4.3 Previous Work

This work builds upon previous work that framed plan selection with preference as an optimal conditional constraint satisfaction problem (OCCSP) [11], by employing dynamic backtracking branch and bound search. Discrete variables in the OCCSP encoded disjunctive choices between methods. In order to frame a looping activity in this way, we would need to encode each loop as a choice. This would effectively produce a TPN as shown Figure 4-5. Instead, we introduced the LTPN as a more compact encoding of the problem (See Figure 4-5), that contains looping episodes to encode conditional looping activities.

Apart from adding looping activities to the problem, we handle preference as a function of the number of loops, rather than a fixed preference value for each disjunctive choice.

Figure 4-5: The encoding of two episodes in a looping temporal plan network as a regular temporal plan network.
4.4 Extracting an Optimal Looping Plan from a LTPN

In order to find an optimal and consistent temporal plan within the LTPN, we introduce LTPN-SEARCH, shown in Algorithm 3. LTPN-SEARCH takes a LTPN as an input and returns the highest utility looping temporal plan (LTP), where choices have been made and loop variables have been fixed to value in their domain. The algorithm systematically searches through the LTPN in an A* manner, using the upper bound on utility as an admissible heuristic to make assignments to decision variables. In the example (Section 4.1), this would correspond to choosing which areas to search and which patterns to perform. Once a complete assignment to decision variables is made, the resulting candidate plan needs to be checked for consistency. If consistent, the optimal number of loops needs to be calculated for each looping episode, as well as the overall utility of the resulting looping plan. To find the optimal number of loops for each looping episode, we frame the problem as a looping temporal problem with preference (LTPP) and solve it as presented in Section 3.4. If this overall utility is higher than any other candidates, the optimal solution is found, otherwise the next highest heuristic candidate is checked for consistency in the next iteration of LTPN-SEARCH.

The algorithm makes use of Candidates to build candidate plans within the LTPN. Each Candidate \(<ADV, UDV, H, LTPP>\) maintains the following information:

- **ADV**: a set of assignments to decision variables, i.e. this set keeps track of decisions made at each decision variable.

- **UDV**: a set of active, unassigned decision variables.

- **H**: a heuristic value. The heuristic value used is the maximum utility obtainable given \(ADV\) (See Figure 4-8).

- **LTPP**: stores the solution to the Looping Temporal Problem with Preference formed during scheduling of a complete candidate. The solution contains all loop variable assignments. \(LTPP\) stores nil if the sub-plan has not been scheduled.
Algorithm 3: LTPN-SEARCH

Input: A looping temporal plan network (LTPN) $< X, DV, E, C >$.

Output: An optimal, consistent looping temporal plan (LTP).

Initialization:
1 $UDV \leftarrow $ activeDecisionVariablesOf(LTPN) ; list of unassigned decision vars
2 $Cand \leftarrow \{\}, UDV, 0, nil$ ; Assigned Events, Unassigned Events, Heuristic and LTPP
3 $priorityQ \leftarrow \{Cand\}$

Algorithm:
4 while $priorityQ \neq \emptyset$ do
5 \hspace{1em} $Cand \leftarrow$ deQueue(priorityQ)
6 \hspace{2em} if isComplete?(Cand) then
7 \hspace{3em} if hasLTPP?(Cand) then
8 \hspace{4em} return formLoopingPlan(Cand, LTPN)
9 \hspace{3em} else
10 \hspace{4em} $CP \leftarrow$ formCandidatePlan(Cand, LTPN)
11 \hspace{4em} $Consistent, Utility, LTPP \leftarrow$ solveLTPP($CP$)
12 \hspace{4em} if Consistent then
13 \hspace{5em} $Cand \leftarrow$ storeLTPP($Cand, LTPP$)
14 \hspace{5em} $Cand \leftarrow$ setCandidateHeuristicTo($Cand, Utility$)
15 \hspace{5em} $priorityQ \leftarrow$ priorityQ $\cup$ Cand
16 \hspace{4em} else
17 \hspace{5em} priorityQ $\leftarrow$ priorityQ $\cup$ expandCandidate($Cand$)
18 return nil

A complete candidate stores a list of assignments ($ADV$) to decision variables ($DX$) that form a complete candidate plan ($CP$) within the looping temporal plan network, i.e. a candidate plan where no activated decision variables are left unassigned ($UDV = \{}$).

Initialization:

LTPN-SEARCH is initialized with candidate $Cand$ on the queue, containing an empty set of assigned decision variables ($ADV$), a set of unassigned decision variables (initially the decision variables in the LTPN that are always active) ($UDV$), a heuristic value of 0 ($H$) and since it has not been scheduled, nil in the $LTPP$ slot. The queue $priorityQ$ contains a list of candidates and is ordered according to the highest heuristic value first. For the search-and-rescue example, the initial candidate
Algorithm:

The algorithm runs until a consistent looping plan is found that has a global utility higher than any other consistent looping plan in the LTPN (line 8), or until the search queue is empty (i.e. there exists no solution) (line 16).

During each iteration of the algorithm, the candidate Cand with the highest heuristic is removed from the queue. LTPN-SEARCH then checks if Cand is complete (Line 6) by checking if the list of active, unassigned decision variables UDV is empty. If Cand is incomplete, it is expanded with all possible assignments an unassigned, active decision variable in UDV (line 15). These child candidates are then added back to the queue (line 15). For example, during the first iteration on the search-and-rescue problem (Figure 4-6), the initial candidate C0 is expanded with the domain of the decision variable AS to two candidates C1 = <ADV : {AS = 2}, UDV : {AE}, H = 54> and C2 = <ADV : {AS = 1}, UDV : {AE}, H = 39>. In the example, the candidates are named in the order they are added to the search queue (See Figure 4-7). The heuristic value is calculated as the sum of the upper-bound on utility obtainable from active episodes and temporal constraints, and the maximum obtainable utility assignments to active, unassigned decision variables (See Figure 4-
For candidate $C1$, the heuristic is 54, since the assignment $A_S = 2$ yields an upper bound on utility of 30, and the maximum utility obtainable by making an assignment to $B_S$ is 24 (Figure 4-6).

Figure 4-7: Search tree for LTPN-SEARCH running on the search-and-rescue mission. Candidates are numbered in the order they are added to the search queue. Complete candidates are shown in grey and the solution is shown in double circles.

At the next iteration, $C1$ is removed from the queue and it is expanded again as variable it is incomplete ( $A_E$ is active but not yet assigned). This process continues until a complete candidate is found (Algorithm 3, line 6 and Figure 4-7, $C5$).

If a candidate $Cand$ is complete and it has not been previously scheduled, then LTPN-SEARCH will use the assignments to decision variables $ADV$ contained in $Cand$ to form a complete candidate plan (line 9). Checking consistency and calculating utility of a candidate plan corresponds to solving a Looping Temporal Problem with Preference (LTPP) (covered in Chapter 3) and is performed in line 10. If $Cand$ is inconsistent, then it is pruned, otherwise the LTPP that contains all assignments to looping variables is stored in the candidate $Cand$. The heuristic value of $Cand$
is changed to the actual maximum utility returned by the LTPP solver and \textit{Cand} is added back to the queue for resorting (lines 12-14). In the example, \textit{C5} is the first complete candidate. After solving the LTPP, it has a utility of 39, thus its heuristic value is changed to 39 and the LTPP is stored.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure.png}
\caption{The figure above shows heuristic values used to guide the search for complete and incomplete candidates. The maximum utility attainable from each episode (i.e. the utility of executing the maximum number of loops) is shown in green. The heuristic of a candidate is the addition of the upper bound of the utility of activated episodes and temporal constraints (in blue), and the maximum utility obtainable from assignments to active, unassigned decision variables (in red). Potentially active episodes and constraints are shown in black.}
\end{figure}

If during any iteration of \textsc{LTPN-Search}, the candidate removed from the queue
is complete and it contains information in the LTPP slot (i.e. the optimal looping parameters have been previously calculated), it is returned as the highest utility, consistent looping plan in the LTPN (line 8). For the search-and-rescue example, the highest utility solution is to assign \( A_S = 2, A_E = 1, B_S = 1 \) to decision variables and assign the looping values calculated in the corresponding LTPP to its looping variables. The optimal looping plan found in the example can be visualized in Figure 4-9.

The looping plan contains a grounded number of loops, but least commitment temporal bounds to reserve some temporal flexibility in when to perform activities. If a looping plan is returned by Algorithm 3, it is guaranteed to be consistent, since it contains a consistent temporal schedule (line 11).

![Figure 4-9: The output of LTPN-SEARCH running on the search-and-rescue problem.](image)

**4.5 Forming a Dispatchable Plan**

The last step to complete our desired capability of this thesis is to compile the optimal looping plan found in Section 4.4 to a dispatchable form. To do this, we expand each executable looping activity in the looping plan into a series of sub-activities, which we call loop flattening. Where a looping activity contains a looping temporal constraint, each sub-activity contains a simple temporal constraint. Figure 4-10 shows a dispatchable plan after loop flattening has been performed on a looping plan.

Algorithm 4 shows the process of loop flattening. The expansion of each looping
TPL (Looping Plan):

```
S \to A_s \to A_k \to B_s \to B_k \to E

\delta_s \in [2, 5]
N_s = 11
```

Dispatchable Plan:

```
S \to A \to \cdots \to A_k \to B \to \cdots \to B_k \to E

\delta_s \in [2, 3]
\delta_A \in [2, 3]
\delta_{AB} \in [3, 4]
\delta_B \in [3, 4]
```

Figure 4-10: A looping plan and a dispatchable temporal plan. Note that the dispatchable plan contains a series of sub-activities rather than a looping activity and it contains tightened temporal bounds and temporal constraints.

**Algorithm 4: LOOPFLATTEN**

**Input:** A looping plan (LTP) < X, A, C >.

**Output:** A dispatchable temporal plan \( \mathcal{P} : < X', SA, C >. 

**Initialization:**

1. \( X' \leftarrow X \)
2. \( SA \leftarrow \{\} \)

**Algorithm**

3. for \( a \in A \) do
   4. \( \text{subactivities, events} \leftarrow \text{expand}(a) \)
   5. \( X' \leftarrow X' \cup \text{events} \)
   6. \( SA \leftarrow SA \cup \text{subactivities} \)
4. return \( \text{APSP}(\mathcal{P}) \)

activity into sub-activities introduces additional events into the plan, as each sub-activity has a start and end event (Definition 11). These are added to the set of events in the looping plan (line 5). The simple temporal constraints in the looping plan are copied to the dispatchable plan. Once all looping activities have been expanded, an all-pairs shortest path is run on the temporal bounds of the plan to form a minimal
network [9]. After this process, the plan may be sent to the Pike executive [20] for online dispatching to a robotic system.

4.6 Chapter Summary

In this chapter, we completed the components needed to dynamically execute a looping temporal plan network (LTPN). We framed the problem of optimal looping plan selection compactly within a LTPN and presented an algorithm LTPN-SEARCH that used an admissible heuristic to form candidate plans within the LTPN by making assignments to decision variables. Then, it called an optimal scheduler SOLVE-LTPP to calculate the loop variable assignments that would result in the highest utility looping plan. Given this, we presented a method for compiling a looping plan into a dispatchable plan.
Chapter 5

Experimental Validation

In this chapter, we first present empirical results for our algorithms for solving the looping temporal problem with preference (LTPP) in Section 5.1, then present empirical results for finding an optimal temporal plan within a looping temporal plan network (LTPN) in Section 5.2.

5.1 Scheduling of Looping Activities

To validate our approach for solving the LTPP, we compared our approach to other candidate methods for solving the problem. The algorithms benchmarked were:

- **SOLVE:LTPP**: Our approach comprised of the two algorithms **DOMAINFILTER** and **BOUNDSEARCH**.

- **COMPSTP**: Comprised of running **DOMAINFILTER**, followed by a best-first component **STP** search. This is similar to the way a **TCSPP** would be solved on this problem instance using preference levels [23].

- **SCiP[2]**: A **MINLP** solver.

For these algorithms, we investigated their run-time and memory requirements as the problem scaled in size. A maximum time of 20 minutes was allowed as we believe this is the maximum allowable time to generate an offline schedule in a realistic search-and-rescue mission.
5.1.1 Experiment Setup

We benchmarked the above methods on LTPPs modelling single vehicle missions, with looping activities interleaved with activities that move between locations or perform some task. The missions are constrained by an overall temporal constraint between the start and end events, as well as other temporal constraints on events in the mission. Figure 5-1 shows the structure of the problem used for benchmarking.

Figure 5-1: LTPP structure for benchmarking.

Since the difficulty of the problem increases with the size of the state-space, we tested performance as we increased the number of looping activities (LTCs). In this case, we constructed the problem with LTCs in series. The LTCs were each of the form: \{N \in [5, 20]; \delta \in [5, 10]; f(N) = a.N\}, where a is a random number between 0 and 5. Simple temporal constraints (STCs) denoting non-looping activities were randomly interleaved between event pairs with probability 0.2. A non-linear global preference function was created by generating a random expression tree, combining pairs expressions using + and \times operators with equal probability. The overall temporal constraint was generated randomly to ensure the solution was varied throughout the state space of possible solutions over multiple runs. Thus, if the number of LTCs in the problem is |LTC|, then, the upper bound of the overall STC was randomly chosen to be between $N_t \times \delta_t \times |LTC|$ and $N_u \times \delta_t \times |LTC|$. 

74
5.1.2 Hypothesis

We expect our approach (SOLVELTPP) to find the optimal solution faster than the best-first, enumeration STP approach (COMPSTP) as our algorithm is designed to search over ranges of loops at a time. The consequence is that although both approaches have the same number of leaves, COMPSTP has a larger branching factor and thus is more shallow than SOLVELTPP. Thus, pruning inconsistencies in SOLVELTPP should eliminate a larger portion of the state-space than pruning inconsistencies in COMPSTP. We also predict a large saving in space required while running, as SOLVELTPP maintains candidates containing ranges of loops on its search queue, whereas COMPSTP maintains candidates of single loop range combinations.

We expect our algorithm to run faster than SCiP as well, since SCiP is designed for a broad range of problems and thus does not leverage the structure of our specific problem. Our algorithms do leverage the structure of the problem, since the domain filtering phase of SOLVELTPP employs constraint propagation to reduce the solution space and the search phase of SOLVELTPP searches over loop ranges, using incremental consistency checking.

5.1.3 Results

Each data point in Figure 5-2 represents the medians of the data for 100 runs. If the median was above the cut-off time of 20 minutes, that data point was not plotted.

Since SOLVELTPP checks consistency over ranges of loops and prunes large parts of the state-space, it shows multiple orders of magnitude improvement in run-time over COMPSTP at 7 looping constraints and above (Figure 5-2). We also show an order of magnitude improvement in run-time over SCiP above 11 looping constraints. This is largely due to efficient constraint propagation to reduce the state space during domain-filtering and tailoring our algorithm to use a monotonic global preference function. It must be mentioned that SCiP can handle non-monotonic functions, which SOLVELTPP does not in its current capacity.

At this setup, the size of the problem is $15^{\lfloor L^{TC} \rfloor}$, where 15 is the number of loop
values in each LTC loop range and $|LTC|$ is the number of looping constraints in the problem. Thus, at a run-time of 600s (10 minutes), SOLVELTPP can solve a problem 2 orders of magnitude larger than SCiP and 5 orders of magnitude larger than COMPSTP.

Although empirical results suggest exponential growth in run-time of SOLVELTPP as the number of looping activities in series increases, SOLVELTPP finds a solution extremely quickly (under 1 second) for problems containing 7 looping constraints or less. This allows it to be used in mobile, online robotic missions of this size.

Another important requirement for algorithms that run on-board mobile robots is that they need to be efficient in the amount of space they require. Thus we also recorded the maximum queue size that SOLVELTPP and COMPSTP maintained while running (See Figure 5-3). Note that the scale of the maximum queue size is logarithmic, thus it can be seen that SOLVELTPP shows an order of magnitude improvement over COMPSTP in the amount of space used during search for 5 LTCs and above. The reason for this is that each candidate stored on the queue for SOLVELTPP contains many possible solutions, while each candidate stored on the queue of COMPSTP stores a single solution.
Figure 5-3: Maximum space requirements of SOLVELTPP vs COMPSTP Search.

5.2 Planning over Looping Activities

To validate our approach for finding an optimal looping plan from a LTPN, we compare LTPN-SEARCH to KIRK [11], an optimal temporal planner, to show the improvement our new addition has made. We investigate the effect of increasing the number of looping episodes in the LTPN, as well as the effect of the number of decision variables in the LTPN.

5.2.1 Experiment Setup

We benchmarked the above methods on LTPNs with many concurrent activities and choices between them, similar to the one introduced in Section 4.1. Figure 5-4 shows the structure of the problem used for benchmarking.

At each temporal event in the LTPN, two looping episodes emerged representing a looping behaviour in the mission, each with a loop range from 5 to 20, a temporal bound per loop of 5 to 10 and a random linear utility function. A decision variable was associated with each event with probability 0.5 (except for the start and end events in the LTPN, which are never associated with decision variables) and had two values in its domain, to represent the two execution threads leaving that event. A conditional
constraint on the time at which events are executed was added with probability 0.5 for each temporal event. These constraints contained a guard that only activated the episode if an assignment to a decision variable caused its end event to form part of the candidate plan. Finally, the missions were constrained by an overall temporal constraint between the start and end event, randomly chosen from a consistent range to ensure the solution was varied throughout the state space of possible solutions.

5.2.2 Hypothesis

We expect the additions made to KIRK, allowing it to reason over looping activities, will make a large improvement in the run-time. Specifically the heuristic used in LTPN-SEARCH to generate candidate plans and the SOLVELTPP sub-routine is expected to be the cause of this improvement.
5.2.3 Varying the Number of Looping Episodes

Since the problem is expected to become more difficult as the number of looping episodes increase, we investigated the effect of increasing the number of looping episode on run time. For each number of looping episodes, LTPN-SEARCH and KIRK were run on 50 problem instances of that size. An upper limit on the run-time of 2 minutes (or 120s in Figure 5-5) was set as search-and-rescue missions are often time-critical and we assumed this to be the limit on the online planning time allowed.

Figure 5-5 shows a large run-time improvement of LTPN-SEARCH over KIRK, allowing many more problems to be solved in the allotted 2 minutes of run-time (Also see Figure 5-6). Note that the dispersion in the results is due to the random variation of the problem solution throughout the solution space. This improvement in run-time is extremely important in time-critical missions (like search-and-rescue and scientific information gathering), where additional planning time may cause a
Figure 5-6: Percentage of problems solved as a function of the number of looping episodes.

loss of information.

Because it is difficult to see how many problems are left unsolved in Figure 5-5, we present Figure 5-6 to show the percentage of solved problem instance as a function of the number of looping episodes in the plan. Above 20 looping episodes in the plan, LTPN-SEARCH solved 40-50% more problems than KIRK. For problems containing 15 looping episodes or less, LTPN-SEARCH solved even the most difficult problem within 2 minutes.

5.2.4 Varying the Number of Decision Variables

The previous section investigated the effect of increasing the number of looping episodes while keeping the probability of each event being associated with a decision variable constant. Now we investigate the effect of varying the probability of each event being associated with a decision variable while keeping the number of looping episodes constant at 30. The set-up of the problem is the same as the previous section, where each decision variable has 2 values in its domain, and each looping episode has 15 values in its loops range domain. The median of the run-time results
for 50 trials at each probability is shown in Figure 5-7.

The Effect of Decision Variables

As can be seen in Figure 5-7, for a fixed number of looping episodes, the problem becomes easier as the number of decision variables increases. In other words, decision variables split the problem into multiple smaller problems to check for consistency. For example, in the experiment above, if there is a decision variable for each event, then the LTPN contains a maximum of 16 candidate plans, each with 4 looping episodes to check consistency over. On the other hand, if there are no decision variables, the LTPN contains 1 candidate plan with 30 looping episodes. Since the scheduling results in Section 5.1 show that the run-time increases exponentially with the size of the problem, solving many smaller problems is faster than solving one bigger problem.

Note that when the probability of decision variables is zero, we are essentially solving a looping temporal problem with preference (LTPP). It is surprising then that LTPN-SEARCH outperforms SOLVE-LTPP in the previous results. This gives us some insight into the effect of problem structure. While the LTPP benchmarking was largely performed on problems with looping temporal constraints (LTCs) in series, the LTPN benchmarking problems had a highly parallel structure. This suggests that optimizing a series composition of LTCs is more difficult than a problem
with a more parallel composition. Figure 5-8 gives some insight into why this is the case by showing two examples in the limit; completely parallel and completely series composition. When all LTCs are in parallel, we can essentially optimize each one separately. However, for LTCs in series, optimizing one affects the others, leading to a multi-dimensional optimization problem. Thus, the more parallel the structure of the problem, the easier it is for our algorithm to solve.

Constraints in Series:

![Diagram showing constraints in series]

Constraints in Parallel:

![Diagram showing constraints in parallel]

Figure 5-8: Looping temporal constraints in series and in parallel. Optimization is more difficult in series as constraints are strongly coupled, while in parallel they are less coupled.
Chapter 6

Conclusion and Future Work

In this chapter, we present interesting directions for future work that were outside the scope of this thesis. Thereafter, we conclude this thesis, by summarizing the key ideas we introduced and the contributions we made.

6.1 Future Work

We suggest two main directions for future work:

1. Increase the speed of optimal plan extraction.

2. Extend optimal scheduling to nested loops.

6.1.1 Increase Speed of Optimal Plan Extraction

Although our algorithm has made a significant advance over previous approaches, both in the run-time and the amount of space used, there are a few areas that could further speed up the search process:

1) Scheduling candidate plans before they are complete:

In this work, we focussed on scheduling candidates after a complete assignment had been made to the decision variables in a LTPN. There may be a benefit to scheduling candidate plans before they are complete, if inconsistencies in the plan are found early. For example, in Figure 6-1, suppose an inconsistency is found after
assigning $A_E = 1$. Then we can prune all decision variables, episodes and temporal constraints that have $A_E = 1$ as their guard.

2) Conflict extraction and resolution:

Another benefit of scheduling incomplete candidate plans is that we might be able to extract a conflict [28] from our assignment to decision variables. Conflict extraction and resolution has the potential to significantly speed up the search. Consider the example in Figure 6-2. Suppose that making the assignment $B = 1$, $C = 1$ yields a negative cycle (temporal inconsistency) during consistency checking between events $A$, $B$, $C$ and $D$. Then we can extract the following conflict: $\{B = 1 \land C = 1\}$ which means $B = 1$ and $C = 1$ cannot both occur at the same time. We can resolve this conflict by using De Morgan’s law: $\neg \{B = 1 \land C = 1\} \Rightarrow \neg \{B = 1\} \lor \neg \{C = 1\}$. In this example, $\neg \{B = 1\} \lor \neg \{C = 1\} \Rightarrow \{B = 2\} \lor \{C = 2\}$. This can be used to guide the search, as we know that when decision variable $B$ and $C$ are both active,
we need to assign either \( B = 2 \) or \( C = 2 \). Allowing these conflicts to guide our search would result in conflict-directed A* [28].

### 6.1.2 Optimal Scheduling of Nested Loops

This thesis handles scheduling of looping activities that contain repeated primitive sub-activities. Suppose that the repeated sub-activities are again looping activities, or compositions of them in series and/or parallel. This introduces the problem of nested loops. An example of this would be picking fruit in agriculture. Suppose we want to pick between 5 and 10 fruit from a tree (inner looping activity), and we want to pick fruit from 10 to 20 trees (outer looping activity), all with an overall duration for the task.

This might be modelled as follows. Suppose we have a looping temporal constraint \( LTC_1 \) that has a repeated looping temporal constraint \( LTC_2 \) as its per-loop duration. Then we have the following form of looping temporal constraints:

\[
\begin{align*}
LTC_1 : & \quad N_1 \in [N_{1l}, N_{1u}] \\
& \quad \delta_1 \in LTC_2 \\
LTC_2 : & \quad N_2 \in [N_{2l}, N_{2u}] \\
& \quad \delta_2 \in [\delta_{2l}, \delta_{2u}]
\end{align*}
\]

The overall temporal bound of \( LTC_1 \) is then: \([N_1 \times N_2 \times \delta_{2l}, N_1 \times N_2 \times \delta_{2u}]\), where
The temporal bounds associated with this problem structure are not too difficult to handle, but the problem becomes more interesting when we consider repeated (series and parallel) compositions of looping temporal constraints and the utility of these nested loops. We believe this is an interesting problem worth pursuing in the future.

6.2 Conclusion

This thesis developed a hierarchical plan executive that quickly finds an optimal temporal plan from a large space of candidate plans comprised of looping activities. This allows our system to be used for time-critical missions such as search-and-rescue. We framed our space of candidate plans succinctly using a looping temporal plan network (or LTPN), an extension of a temporal plan network, which compactly encodes multiple candidate solutions. Our goal was to extract an optimal, feasible and temporally-flexible plan by choosing between sub plans and by selecting loop parameters. This was represented by finding a temporally consistent assignment to the decision variables and loop variables in LTPN, so as to maximize the overall utility of the plan. We returned a temporal plan containing those choices, along with consistent temporal bounds for constraints and loop durations so as to ensure a consistent schedule exists.

Our approach to solving the problem was to extend optimal temporally-flexible planning methods to include reasoning over looping activities to find an optimal plan from many candidates. The approach contains three main parts:

1. **Candidate Generator**: The first step was to generate a complete candidate by searching through the hierarchical LTPN in an A*-like manner, making assignments to decision variables. We used the upper bound on utility as an admissible heuristic to make these assignments. In the search-and-rescue example we introduced, this corresponded to choosing which areas to search and which search patterns to perform.
2. Optimal Scheduler: Once a complete candidate was found, it needed to be checked for consistency. If consistent, the optimal number of loops and consistent temporal bounds needed to be calculated, as well as the utility of the candidate. This was done by first employing a domain filtering technique to prune inconsistent loops and then using a best-first search technique to calculate the optimal loop assignments and the overall utility of the candidate.

3. Plan Reformulation: After finding the optimal looping plan, we reformulated the plan through a process of loop flattening into a form that could be dispatched by an online dispatcher.

To enable fast and optimal temporal planning over looping activities, this thesis made the following key contributions, required to speed up the search and reduce the number of candidate plans searched through before finding the optimal:

- We framed optimal plan selection within a looping temporal plan network (LTPN) (Chapter 2).

- We introduced an optimal heuristic search to find an optimal and consistent solution to the LTPN (Chapter 4).

- We introduced a new problem, the looping temporal problem with preference (LTPP) to encode temporal problems that contain looping activities (Chapter 3). The LTPP formed a sub-problem to finding an optimal looping plan in the LTPN.

- We solved the LTPP by introducing a novel constraint propagation algorithm to reduce the space of possible solutions and a heuristic best-first search algorithm that searches over ranges of loops and checks consistency incrementally to find an optimal solution efficiently (Chapter 3).

Our capability allows robots to find optimal parameters within search patterns in order to maximize the information gain of their search strategies. The improvement in run-time and space requirements of our algorithm over previous work allows it to
be run on processors on-board real robotic systems and replan online. Given this, we make a large step towards robotic scouts that can autonomously collect data and search for missing people or hijacked vehicles, while replanning and adjusting the plan based on information gained during the mission.
Bibliography


